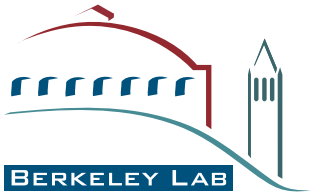
FH Karlsruhe
UNIVERSITY OF APPLIED SCIENCES
HOCHSCHULE FÜR TECHNIK



Design and implementation of an FPGA based Digital Correction and Calibration system for a high-resolution low power pipeline ADC for space applications

by
Markus Redelstab
Diplomarbeit Fachbereich EIT Nachrichtentechnik
FH Karlsruhe – University of Applied Sciences
Prof. Dr. Bantel, Prof. Dr. Sapotta
developed at Lawrence Berkeley Lab
2004/2005

ABSTRACT

CRIC (**C**CD **R**eadout **IC**) is a 13bit, four channel, low noise, low power pipeline ADC for CCD applications. To process readout data and to get relevant performance benchmarks, a measurement setup had to be developed that is flexible in design and capable of providing the following functions:

- Pattern generation and timing of all internal and external switches during ADC operation within a fixed timing scheme of 10µs-cycles with 400 time slots of 25ns, i.e. f_{oper} = 40MHz
- Simultaneous acquisition of measurement data from all four channels
- Deserializing of ADC output (4×LVDS input to FPGA)
- UART interface for communication with a serial terminal
- Receiving of commands to adjust and control CRIC
- Sending of measurement related information, like calibration constants
- Digital Correction and Calibration in real-time
- Output of processed ADC data on a time multiplexed 32bit bus

Since most of the mentioned operations rely on certain timing schemes and since part of the signals is LVDS, the use of an FPGA is the ideal solution to perform these tasks. Thus all functions are realized on a Xilinx Spartan II-E with 200kGates, which meets all of the above requirements. The FPGA board is attached as a daughterboard to the CRIC test circuit via two 140pin high-density connectors. The daughterboard is connected with a serial cable to a PC running special test software to control CRIC during operation, while a PCI I/O card receives ADC output data and sets onboard DACs when testing the ADC inputs. The core function of the design is to apply Digital Correction and Calibration to all measurements which was initially done by the test software too. Furthermore it was used to gain measurement data like noise, linearity (DNL / INL) and crosstalk of CRIC. The proposed VHDL design is supposed to be implemented hard-wired and on-chip in the next CRIC version.

Table of Contents

1 Introduction....6
1.1 SNAP – SuperNova / Acceleration Probe....6
1.2 CCD....7
1.3 CRIC-II overview....8
1.4 Core specifications....9
2 Analog signal processing....11
2.1 Noise sources....11
2.1.1 Output amplifier noise....11
2.1.2 Reset noise....12
2.1.3 Shot Noise....12
2.1.4 Conclusion....12
2.2 Correlated Double Sampling (CDS)....13
3 Typical ADC architectures....14
3.1 Comparison of current ADC technologies....14
3.2 ADC speed versus resolution....15
3.3 Advantages of Pipeline ADCs....15
4 Pipeline ADCs....16
4.1 The ideal Pipeline ADC....16
4.1.1 Architecture....16
4.1.2 The 1bit cell....17
4.2 The non-ideal Pipeline ADC....17
4.2.1 Conversion errors....17
4.2.2 Architecture....18
4.2.3 Correction logic....19
4.2.4 Conclusion....19
4.3 CRIC-II digital block....20
4.3.1 Architecture....20
4.3.2 The 1.5bit cell....20
4.3.3 Digital correction logic....21
5 Digital error correction methods....22
5.1 Digital Correction of 1.5bit cells in a 13bit ADC....22
5.2 Digital Calibration of 1.5bit cells in a 13bit ADC....22
5.2.1 Theory....22
5.2.2 Calibration process of CRIC-II....24
6 Hardware setup....27
6.1 CRIC input and output signals....27
6.1.1 Pin assignment....27
6.1.2 Signal standards....30
6.2 CRIC-II test board....31

Table of Contents

6.3 FPGA board....31
7 FPGA design implementation....33
7.1 Design overview....33
7.2 Timing diagrams....34
7.3 Design considerations....37
7.3.1 Pattern generation:....37
7.3.2 Storage of calibration constants....37
7.4 State machines....38
7.5 Subcomponents....38
7.6 State charts....39
7.6.1 Overview....39
7.6.2 Main state machine (SST)....40
7.6.3 Pattern state machine (PST)....48
7.6.4 Configuration state machine (CST)....48
7.6.5 Calculation of calibration constants state machine (DST)....49
7.6.6 FST: Final value state machine:....50
7.7 Design process and synthesis with Xilinx ISE....51
8 Measurements....52
8.1 ADC specifications....52
8.1.1 INL: Integral Non-linearity....52
8.1.2 DNL: Differential Non-linearity....53
8.2 INL measurements....54
8.3 DNL measurement....55
8.4 Noise measurement....56
8.5 Statistical evaluation of the calibration constants....57
9 Conclusion and future upgrades....58
9.1 Meeting the specs....58
9.2 The way to CRIC-III....58
10 Appendix....59
10.1 VHDL code documentation....59
10.1.1 Pattern generation – Calibration – Main Control....59
10.1.2 ADC Digital Correction....69
10.1.3 Binary to Hex Transcoder....70
10.1.4 RAM 256x16....71
10.1.5 RAM 512x16....72
10.1.6 Mini UART....73
10.2 Example: Synthesis of Digital Correction....74
10.3 List of figures....76
10.4 List of Tables....78
10.5 Further information....78
10.6 Glossary....79

10.7 Acknowledgments.....82
10.8 References.....83

1 Introduction

1.1 SNAP – SuperNova / Acceleration Probe

By observing distant, ancient exploding stars it is possible to determine that the universe is expanding at an accelerating rate. Right now, the most reasonable explanation for these observations is the existence of a mysterious, self-repelling property of space first proposed by Albert Einstein, which he called the cosmological constant. This surprising discovery that the expansion of the universe is accelerating, and therefore is likely to expand forever, is based on observations of type 1a supernovae, very bright astronomical "standard candles" that all have the same intrinsic brightness. With these exploding stars you can derive the distance from their measured brightness and by comparing the distance of these exploding stars with the red shifts of their home galaxies, you can calculate how fast the universe was expanding at different times in its history. Good results depend upon observing many type 1a supernovae, both near and far.

The recent strategy to find these supernovae is to make images of 50 to 100 patches of sky, each containing roughly a thousand distant galaxies. Three weeks later the same patches are imaged again. Supernovae occurring anywhere in these fields show up as bright points of light. This procedure was developed in the 1990s by Supernova Cosmology Project member Gerson Goldhaber and is called "supernovae on demand". But to be able to do further and more precise research on this subject, it would be very useful to find more supernovae so as to avoid systematic errors providing the opportunity to examine them more precisely.

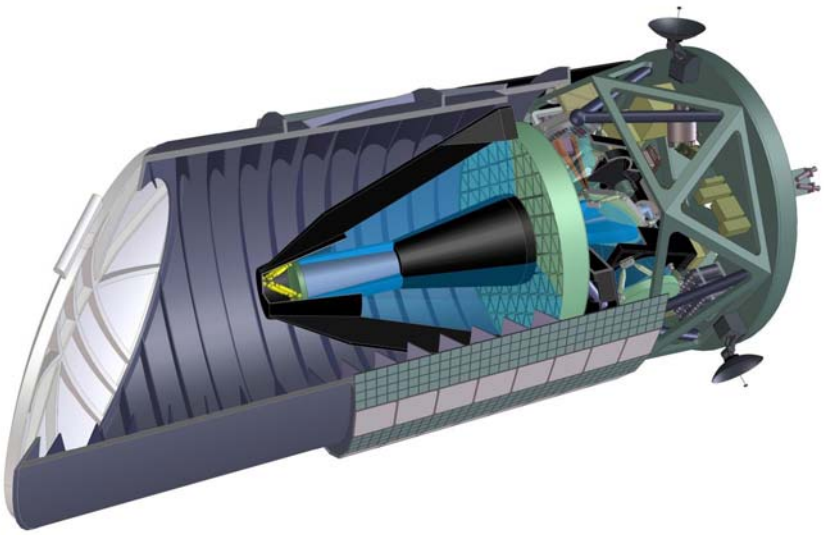


Figure 1: SNAP satellite

That’s why a space mission with a satellite is now being considered that would increase the discovery rate for such supernovas to about 2000 per year. This satellite called SNAP (Supernova / Acceleration Probe) would be a space-based telescope with a one square degree field of view with 1 billion pixels. This implies exceptional CCDs that are very sensitive over a wide range of wavelength, as well as readout electronics that have features like low noise and low power together with high resolution and support for highly integrated circuit design.

1.2 CCD

CCDs (Charge Coupled Devices) are silicon based and highly sensitive light detectors and were first conceived in 1970 in the Bell Labs. They've got a large number of small light sensitive areas (pixels) that convert photons to electrons. Whenever a photon hits a pixel, there will be -depending on the energy- one or more electron-hole pairs. The potential in the silicon is shaped in a way that these electrons get captured, just like in a bucket. So after a certain exposure time the number of electrons collected will be directly proportional to the intensity of the scene at each pixel.

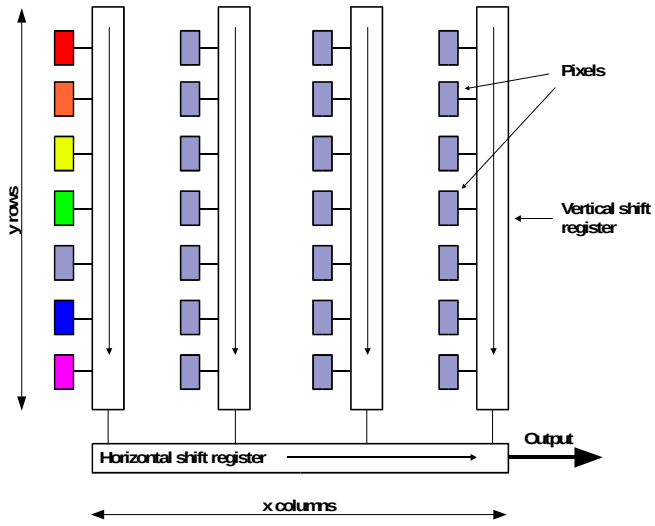


Figure 2: CCD shift registers

Each column is connected to a vertical shift register *x*, that is connected to an horizontal shift register *y* as well *(see figure 2)*. If you now attach a clock signal to all columns, the charge will begin to move down from one cell to the next and from the last cell at the bottom into the horizontal shift register, just like in a bucket chain *(figure 3)*. So each clock cycle for reading one row is followed by *x* clock cycles to readout the horizontal shift register. Finally, after repeating this procedure *y* times, the entire CCD is processed and the image scene can be reconstructed.

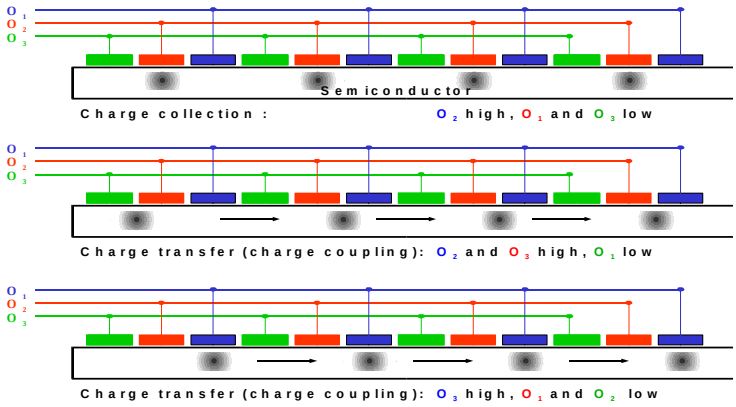


Figure 3: CCD charge coupling

1.3 CRIC-II overview

The core functions to readout the CCD are provided by CRIC-II (CCD Readout IC) ASIC developed by the LBL IC design group. It comprises the following circuit blocks:

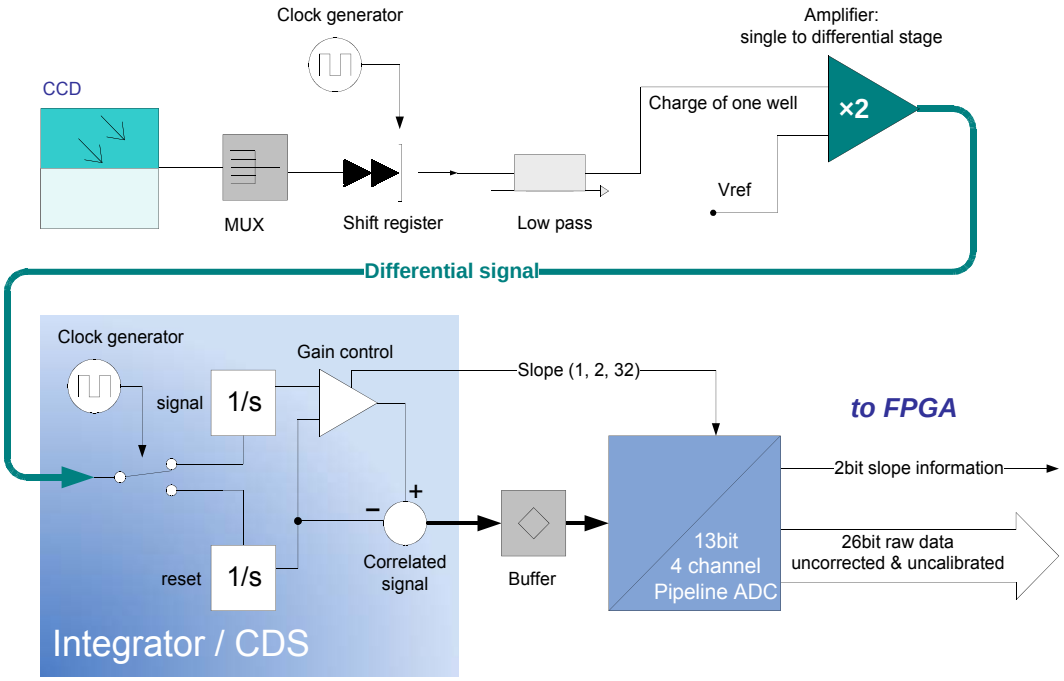


Figure 4: CRIC-II overview

As explained in the preceding chapter, the CCD is read out by selecting a column (MUX) and by clocking out the charge of all CCD wells into shift registers. Inside the amplifier the charge is transformed via a sense capacitor C_S into its corresponding voltage ($U=Q/C_S$). Prior to the A/D conversion the signal has to be low-pass filtered to toss all frequency components outside Nyquist zone. Otherwise any (unwanted) frequency component would be aliased back into the target zone and decrease SNR performance. The next stage represented by a CDS circuit (Correlated Double Sampler) is used to reduce readout noise by correlating the signal with its reset level. Depending on the input voltage the amplifier switches between several slope rates to allow a good dynamic range. The final output is 4 channels × 26bit of uncorrected and uncalibrated data, which has to be processed afterwards by a correction circuit inside the FPGA to output 13bit effective data resolution.

On the next page you can see a photo of CRIC-II, which gives an idea of the four channel symmetric block design:

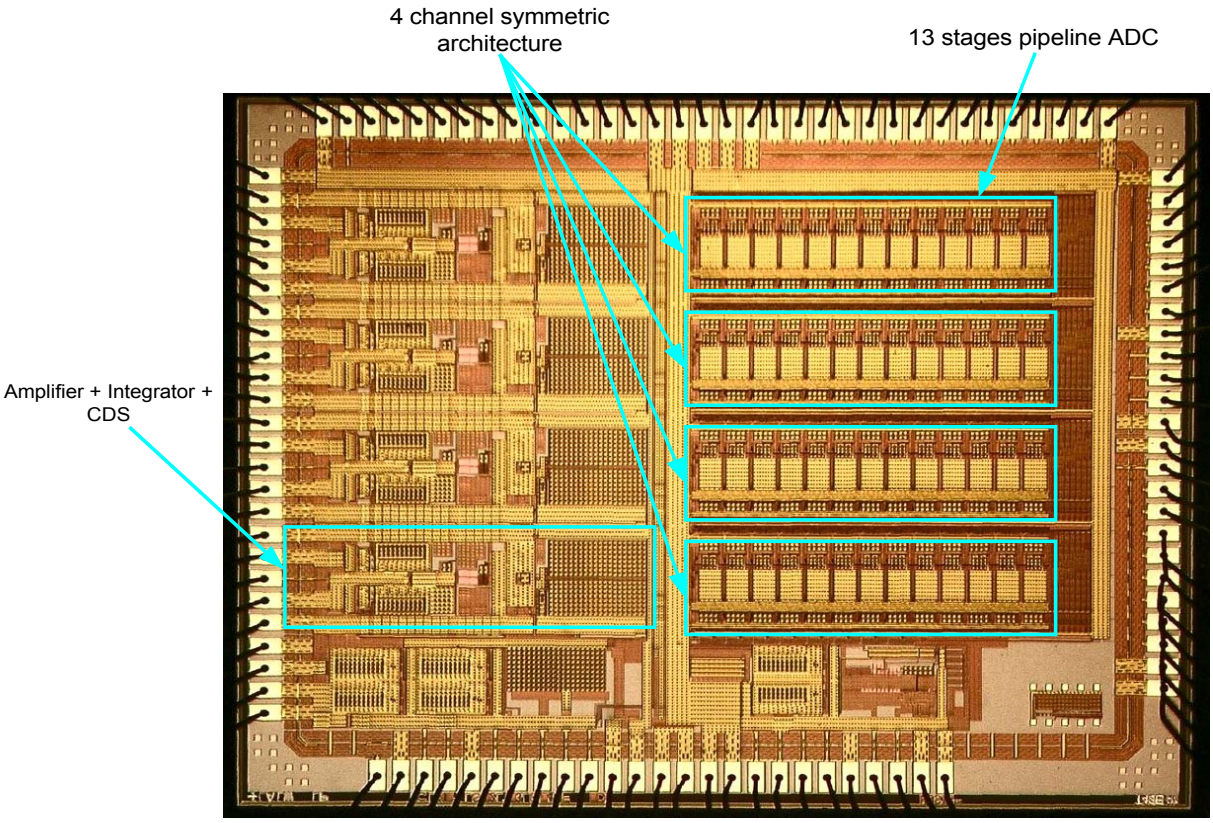


Figure 5: CRCIC-II photo

1.4 Core specifications

Input amplifier:

- Gain: 2
- High input impedance
- Input voltage swing ±500mV
- Supply voltage: 3.3V
- Bandwidth >> 3.5MHz

Integrator:

- Covers full dynamic range: adjustable gain of 32, 2, 1
- Input voltage swing ±1V

CDS:

- CCD + CDS read noise: $4e^-$ @ 100 kS/s, goal of $2e^-$ @ 50 kS/s
- Pixel rate: 100 kS/s (50 kS/s for spectrograph) -10% readout dead time

ADC:

- 13bit resolution
- 13 pipelined stages
- 1.5bit/stage
- INL: ±0.5 LSB (after calibration)
- DNL: ±0.5 LSB (after calibration)
- ADC noise: 1 LSB
- Inter-stage gain: Slightly above two
- Thermometer to Binary transcoder
- Digital Correction
- Digital Calibration of first 8 MSB cells
- Fully differential
- Switch OTA architecture
- 4V full scale
- Large dynamic range: 96dB from noise floor to 130ke well depth
- Readout rate:100 kHz

Common requirements:

- Operation at 140K and room temperature
- Radiation tolerance: 100 krad TID (20 krad shielded by focal plane)
- Low power: mandatory for space application in general, very important for readout electronics due to placement close to CCD
- Small footprint
- Highly integratable

2 Analog signal processing

2.1 Noise sources

Beside intrinsic CCD noise already implied in the CCD signal, there are several sources of noise in the readout electronics. Specifically they are:

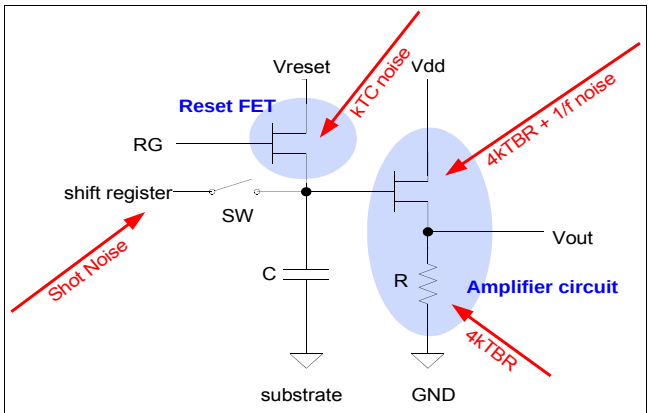


Figure 6: Noise sources

2.1.1 Output amplifier noise

White noise, also well known as Johnson noise which is determined by the Nyquist formula:

$\sigma_{Johnson} = \sqrt{4\,kTBR}$, where

- k: Boltzmann's constant [J/K]
- T: temperature [K]
- B: noise equivalent bandwidth [Hz] determined by the 3dB bandwidth of the amplifier's transfer function
- R: output impedance of amplifier (1/gm for source follower)

This type of noise is usually Gaussian, additive and independent of the signal and occurs in any resistor.

Flicker noise, also called 1/f noise since it is inversely proportional to frequency, is basically produced by any kind of metal junction, i.e. in all MOSFETs of the amplifier. As you can see in the following diagram, noise decreases as both temperature drops and frequency rises.

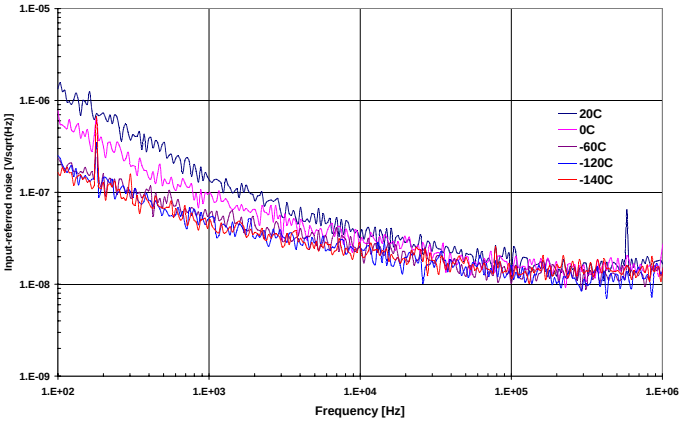


Figure 7: Flicker noise

2.1.2 Reset noise

Whenever a charge is translated to a voltage, there's an uncertainty due to noise generated by the channel resistance of the reset FET. This noise is also defined by the Nyquist formula, but usually expressed in terms of electrons to compare it with the amount of electrons in the wells of the CCD:

$$\sigma_{reset} = \sqrt{4kTBR} \qquad \text{and} \qquad B = \frac{\pi}{2} \cdot f_0 = \frac{1}{4RC} \quad \Rightarrow$$

$$\sigma_{reset} = \sqrt{\frac{kT}{C}} \qquad \text{and} \qquad Q = n \cdot e = C \cdot \sigma \quad \Leftrightarrow \quad \sigma = \frac{n \cdot e}{C} \quad \Rightarrow$$

$$n_e = \frac{C}{e} \cdot \sqrt{\frac{kT}{C}} = \frac{\sqrt{kTC}}{e}$$

That's why Reset Noise is also often called kTC noise.

2.1.3 Shot Noise

Due to the unpredictability of time between two arrivals of electrons on a detector, this kind of noise represents the absolute minimum noise level that could be achieved if any other noise source could be eliminated. The time between two arrivals of photons is Poisson distributed:

$\sigma = \sqrt{S}$, where

S is the signal in electrons and
σ means the noise in electrons resulting from this exposure.

2.1.4 Conclusion

However, the main noise sources on the CCD itself are photon noise and dark current. They're strongly influenced by architecture and quality of the CCD on one hand and temperature on the other hand. Optimization of architecture is subject of current research and will achieve further noise reduction. Furthermore the discussed CCD application is supposed to work at low temperature (140K), so particularly dark current is extensively reduced.

2.2 Correlated Double Sampling (CDS)

Read-out noise increases in proportion to read-out speed. The cost of going faster is more noise and hence, more uncertainty in the voltage determination and lower number of bits of resolution. So using a sample rate of 100kS/s is a compromise between operating at moderate speed and having good accuracy by lowering the noise floor. To eliminate most parts of this noise, a circuit called Correlated Double Sampler (CDS) is widely used. The name is derived from the procedure applied to get the final (noise reduced) signal which correlates two samples using the described time pattern. The following picture shows an oscilloscope screenshot of the CCD readout signal during one sampling period:

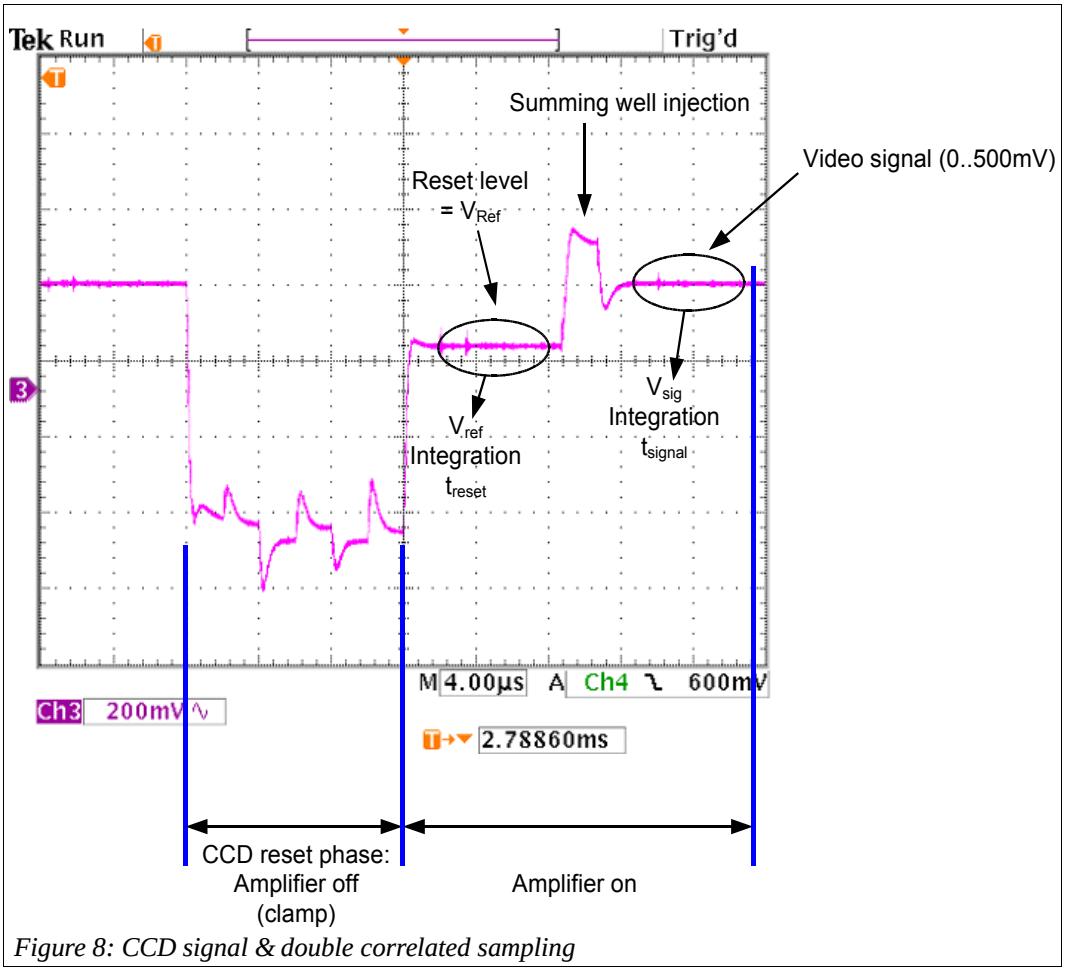


Figure 8: CCD signal & double correlated sampling

The first integration sample (Δt_{reset}) is acquired to $V_{ref} = Q_{ref}/C$ at the end of the reset phase (see ellipsoid integration area), whereas the second sample $V_{signal}=Q_{pixel}/C$ is taken while the CCD signal is switched active. Ideally, the two samples differ only by a voltage corresponding to the transferred charge signal. This is the video level minus the noise $V_{out} = V_{signal}-V_{ref} = (Q_{pixel}-Q_{ref})/C$. The CDS function will eliminate two major sources of noise, as they are kTC noise and part of the White noise.

3 Typical ADC architectures

3.1 Comparison of current ADC technologies

To get a better understanding of ADCs, it is useful to know about the basic differences of the major designs currently competing. The following table opposes their main characteristics and demonstrates that each design has its benefits and disadvantages and makes it more or less suitable for a particular application.

	Flash	***SAR*** *(Successive Approximation Register)*	***Dual Slope*** *(integrating)*	***Pipeline***	***Sigma Delta***
Main features	Sample rate up to ultra-high speed	Medium to high resolution (8 to 16bit), 5MS/s and below, low power, small size	High resolution, low power, good noise performance	High speed, up to 100+ MS/s, 8bit to 16bit, low power, low noise	High resolution, low to medium speed
Conversion method	n-bit lead to 2^{n-1} comparators, # of capacities increase by a factor of 2 for each bit	Binary search algorithm, internal circuit runs higher speed	Unknown input voltage is integrated and value compared against known reference value	Small parallel structure, each stage works on one to a few bits	Oversampling ADC, 5-Hz - 60Hz rejection, programmable data output
Encoding method	Thermometer Code Encoding	Successive Approximation	Analog Integration	Shift register, Digital Error Correction logic (DEC)	Oversampling modulator, digital decimation filter
Dis-advantages	Metastability (sparkle codes), high power consumption, large size, expensive	Speed limited to ~5MS/s, may require antialiasing filter	Slow conv. rate, high precision ext. components required to achieve max. accuracy	Parallel structure increases throughput at the expense of power and data latency	Higher order n-bit ADC and n-bit feedback DAC, may require anti-aliasing filter
Conversion speed	Conversion time does not change with increased resolution	Increases linearly with increased resolution	Conversion time doubles with every additional bit of resolution	Increases linearly with increased resolution	Tradeoff between data output rate and “noise free“ resolution
Size	2^{n-1}comparator, die size and power increases exponentially with resolution	Die size increases linearly with increase in resolution	Die size will not materially change with increase of resolution	Die size increases linearly with increase of resolution	Die size will not materially change with increase of resolution

Table 1: ADC comparison

3.2 ADC speed versus resolution

The diagram below illustrates the tradeoff between speed and resolution and shows the typical operation areas of the major ADC architectures.

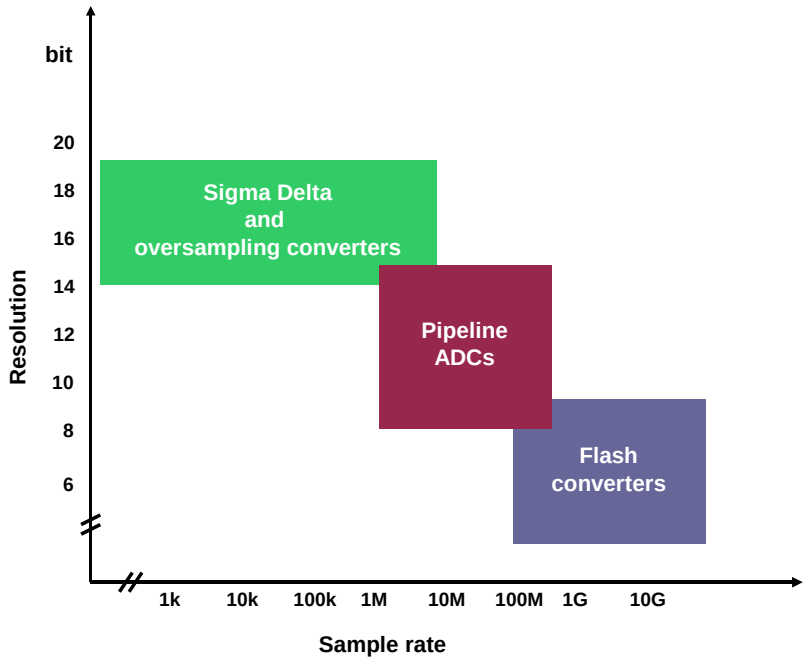


Figure 9: ADC speed vs. resolution

3.3 Advantages of Pipeline ADCs

As you have seen in the chapters before, Pipeline ADCs offer a very good balance of speed, resolution, size and power dissipation. Thus, the Pipeline structure covers a wide range of applications and is still the most popular design, even though it was invented in 1964 by T.C. Vester and isn't the latest ADC design. Among many other applications, CCD imaging is the fastest growing segment, particularly driven by the increasing demand for imaging systems, e.g. photo cameras. The available sampling rates vary from a few mega samples to more than 500 MS/s, which is the top speed segment among ADC designs and is only outperformed significantly by Flash ADCs.

The most important advantages of Pipeline ADCs over the other competing designs mentioned before are the very good noise performance and the low feedback from the digital output to the (analog) ADC input. This is given by the ADC's timing scheme that has digital transitions occurring only while the analog front-end is switched off. That's crucial since any open back path would affect the input signal.

4 Pipeline ADCs

4.1 The ideal Pipeline ADC

To get into the basic concept of Pipeline ADCs it is useful to have a closer look at the Pipeline structure in an ideal environment, i.e. without noise and offset. This way the ADC design becomes fairly straightforward because there's no need for circuits additional to the conversion electronics to handle these undesirable effects.

4.1.1 Architecture

The following diagram shows a k-stage Pipeline ADC with 1 bit resolution per stage and k subranging cells:

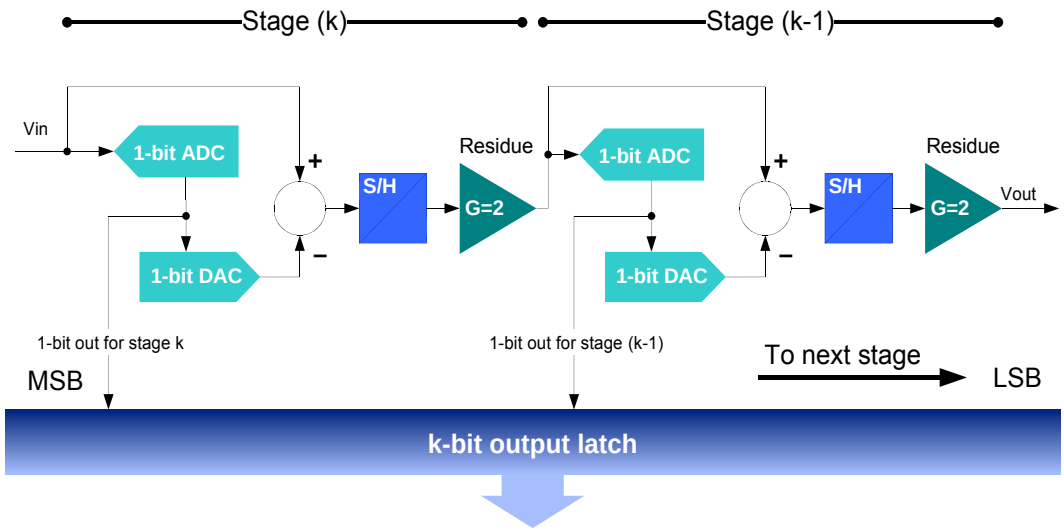
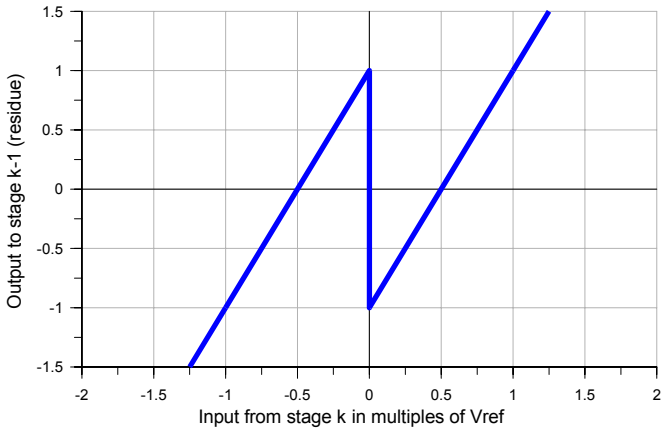


Figure 10: Pipeline ADC with 1bit per stage

Beginning with the MSB the analog input is sampled by a 1-bit (Flash-) ADC that gives the result for stage k and is converted back by a DAC to calculate the residue for the next stage (k-1). The result of this (analog) subtraction is transferred to a Sample&Hold circuit from where it gets amplified (G=2) to refine the result by passing it to the next stage. That's where the term pipelined architecture is derived from. It refers to the ability to process the data from the previous stage, i.e. the data is transferred through the cells like in a pipeline. This procedure continues to the last stage which examines the final residue and represents the LSB. Since each cell provides 1 bit, the output logic is 100% transparent and simply comprises k delay elements with paths of different length to synchronize all bits and to compose the final conversion result.

4.1.2 The 1bit cell

The ideal transfer function for a 1bit cell is shown below:



$$\mathbf{1} \rightarrow V_{out} = 2 \cdot V_{in} - V_{ref}$$

$$\mathbf{0} \rightarrow V_{out} = 2 \cdot V_{in} + V_{ref}$$

Figure 11: Transfer function 1bit cell

After the binary decision is made (0=left negative side, 1=right side), the residue is passed to the next cell to refine the conversion result by a factor of two and thus, to add an additional bit of resolution. So if input values around U=0V are applied, i.e. if it is very hard to make a decision to “0” or “1” and the associated uncertainty becomes ultimate, the residue will be close to the conceivable maximum too. Unless there's no comparator offset and the inter-stage gain is exactly G=2, the ADC operates smoothly. However, in real applications the ADC circuit should be expected to have some offset and gain variations. This has certain consequences and tradeoffs for the design, that will be discussed in the next chapter.

4.2 The non-ideal Pipeline ADC

4.2.1 Conversion errors

In first line ADC designers have to face offset and gain error due to charge injection and capacitance mismatch, which occurs inside the comparators and amplifiers when switching their inputs or during charge transfer respectively. That's an issue because if the residue of

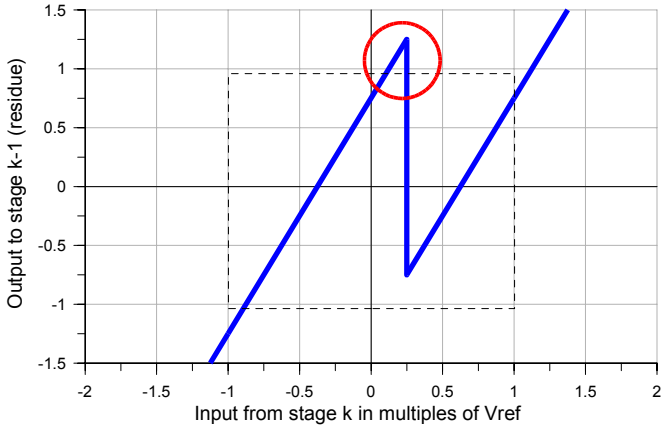


*Figure 12: Transfer function 1bit cell * offset*

stage k is close to the maximum input rating of stage (k-1) the additional offset voltage or gain misalignment will saturate the stage. This effect is shown in *figure 12*.

Whenever the transfer function leaves the indicated area there's a loss of information because everything outside is clipped off. The consequence for the ADC operation are jumps in the quantization function which ultimately lead to missing codes on the digital output that are absolutely fatal to any ADC design*(see figure 13)*.

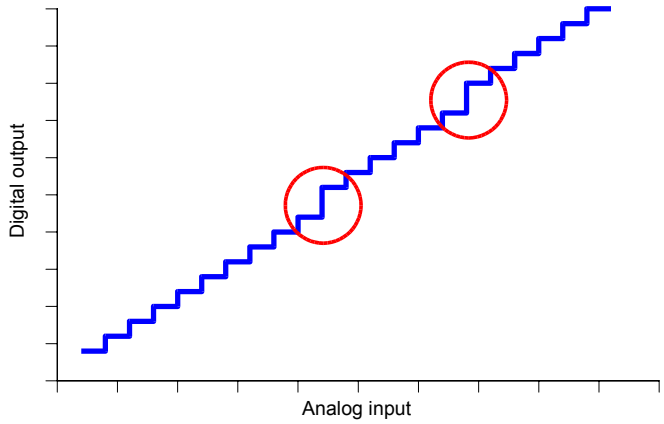


Figure 13: ADC missing codes

So there are two choices to eliminate this problem: To either prevent the occurrence of offset and gain error (which is very expensive and hard to realize) or to make it tolerant to these type of errors (which is more reasonable).

4.2.2 Architecture

The common approach to correct the mentioned errors is to add some redundancy to deter saturation and thus loss of information. This is achieved by adding one or more comparators to each cell to enlarge the dynamic range. *Figure 14* illustrates these modifications.

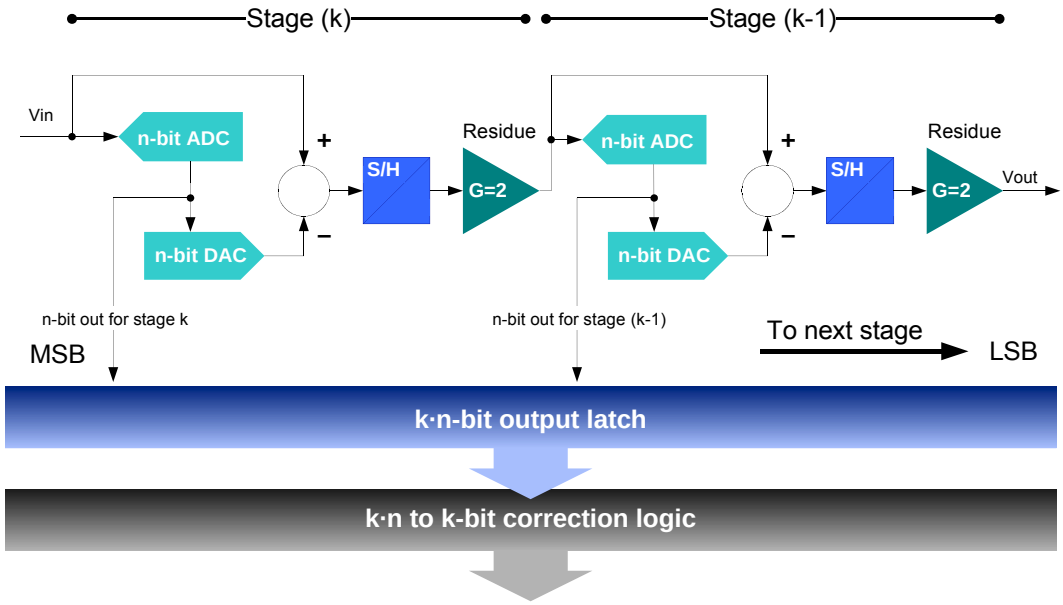
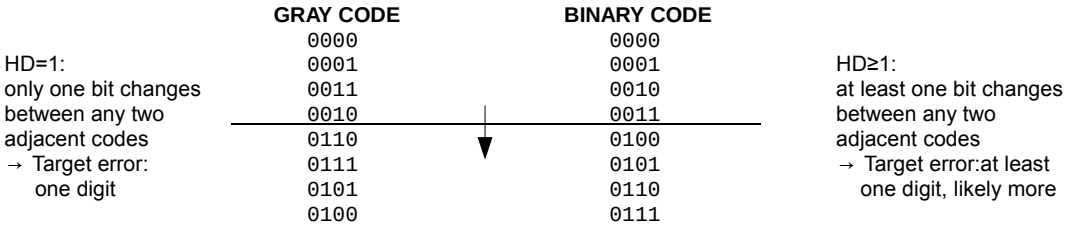


Figure 14: structure n-bit ADC

4.2.3 Correction logic

By adding redundancy with extra comparators, the output of each stage doesn't necessarily represent a binary code any more. Depending on the applied coding method and according to the number of bits per stage (n-bit ADC/DAC), the output has to be processed by a correction logic to get a valid binary representation of the analog input. This involves two steps: First, the intermediate code delivered by each stage, which is usually encoded using codes with a small Hamming distance to avoid large errors (Gray, Thermometer, etc.) is translated to a binary representation. Second, this binary representation is digitally corrected to get a binary number as result of the entire A/D conversion. (see chapter 6.1)



Further improvements can be achieved by using a method called Digital Calibration. That's one of the core tasks of the FPGA implementation. Hierarchically it is a subsequent operation layer right after the Digital Correction. (see chapter 6.2)

4.2.4 Conclusion

Extra comparators bring redundancy and more dynamic range. The consequence is an overlapping of the dynamic ranges, which prevents the occurrence of saturation due to offset errors. So there's no loss of information any more. This is realized to the expense of extra logic because the intermediate cell code has to be translated to a valid binary code. The method of Digital Calibration is capable of achieving further error reduction and can be added by implementing extra computation logic. After applying this last step of code correction all redundancy is used and therefore removed. It has to be emphasized that only offset error can be corrected, whereas negative gain error ($G<2$) cannot be corrected at all and has to be avoided under all circumstances. Positive gain variation ($G>2$) can be tolerated to the expense of slightly smaller residue range, i.e. for this case the maximum (V_{ref}) isn't reached any more.

4.3 CRIC-II digital block

The CCD as part of the imaging system is read out by a digitally corrected and calibrated 13bit Pipeline ADC with 1.5bit per stage. The following chapter describes this specific architecture.

4.3.1 Architecture

Below you can see the final structure that was used to process all ADC output data:

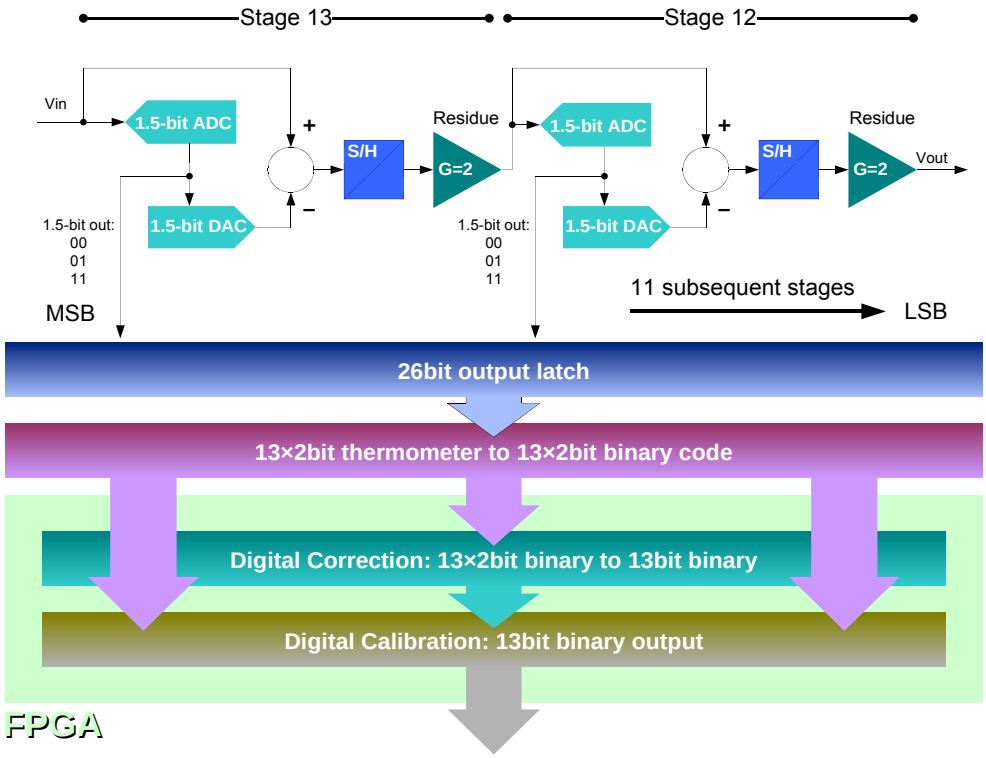


Figure 15: CRIC-II 13bit Pipeline structure

Now, the error correction layer gains more importance since the extra redundancy has to be processed by additional logic. Thus, in addition to *figure 14* this layer is split in three sub-layers, the first (Therm2Bin, *see 4.3.3*) is part of the CRIC design the last two (Digital Correction and Calibration) are implemented in an FPGA.

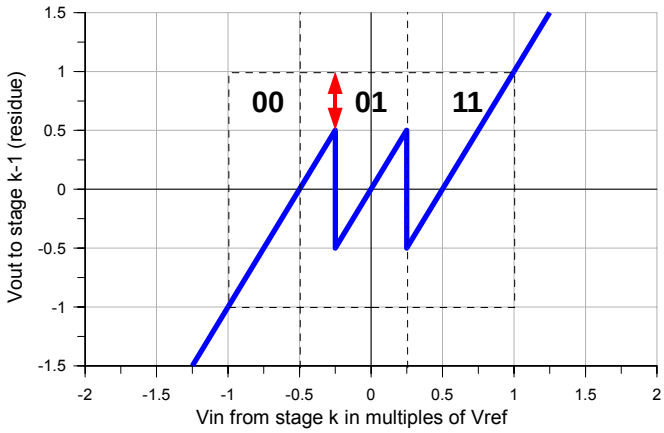
4.3.2 The 1.5bit cell

The use of 1.5bit cells allows overlapping ranges and enough redundancy to circumvent missing codes. The 1.5bit cell has three states (Thermometer code: 00,01,11) represented by two comparators. So the term 1.5bit cell refers to the redundant usage of two comparators that don't represent two, but 1.5bit:

For n comparators there are: $a = 2^n$ binary combinations, ergo it represents $ld(a)$ bits:

$a = 4$ combinations $\rightarrow$ $ld(4) = 2$bit
$a = 3$ combinations $\rightarrow$ $ld(3) \approx 1.5$bit

The figure below shows the transfer function of a 1.5bit cell:



$$\mathbf{00} \rightarrow V_{out} = 2 \cdot V_{in} + V_{ref}$$

$$\mathbf{01} \rightarrow V_{out} = 2 \cdot V_{in} + 0$$

$$\mathbf{11} \rightarrow V_{out} = 2 \cdot V_{in} - V_{ref}$$

Figure 16: Transfer function 1.5bit cell

As indicated in *figure 16* there's much more headroom for inter-stage misalignment now. Offset and gain error can be as large as $0.5V_{ref}$ before the stage exceeds the safe operation area.

4.3.3 Digital correction logic

As mentioned the correction logic includes three layers, the first (Therm2Bin) is part of the CRIC design and is explained below. The layers underneath are not part of CRIC's digital block and are realized in an FPGA.

Thermometer to binary transcoder

Before the code coming out of the stages is passed to the Digital Correction, it has to be translated to a binary representation. The following table shows the allocation:

Thermometer	*Binary*
00	00
01	01
11	10

Table 2: Therm2Bin transcoding

Digital Correction and Calibration

Since Digital Correction and Calibration are both part of the FPGA implementation a detailed functional description is given in the next chapter.

5 Digital error correction methods

5.1 Digital Correction of 1.5bit cells in a 13bit ADC

The Digital Correction works as follows:

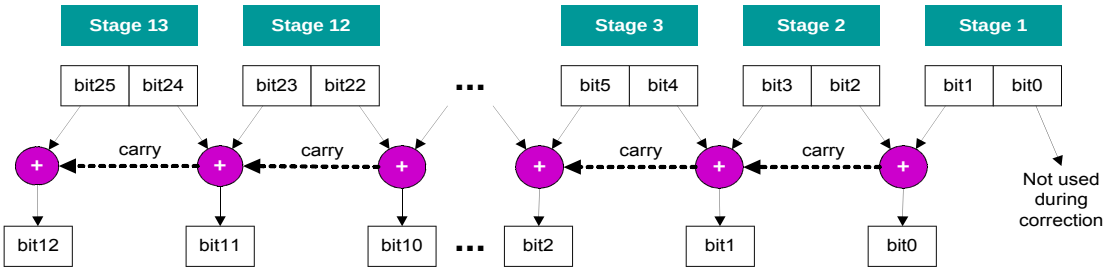


Figure 17: Digital Correction

Beginning with the MSB part of the first stage (bit1), the inter-stage summing starts from right to the left. As result of the first summing operation a Carry bit is passed upward and represents one of three inputs of the next cell's XOR gate. This process propagates up to the last stage, so each gate output forms one bit of the final result.
Fore more details see *Appendix 10.2: Synthesis of Digital Correction*

5.2 Digital Calibration of 1.5bit cells in a 13bit ADC

5.2.1 Theory

Ideally, after Digital Correction and before Calibration with an inter-stage gain of exactly G=2 there's the following situation at each stage output:

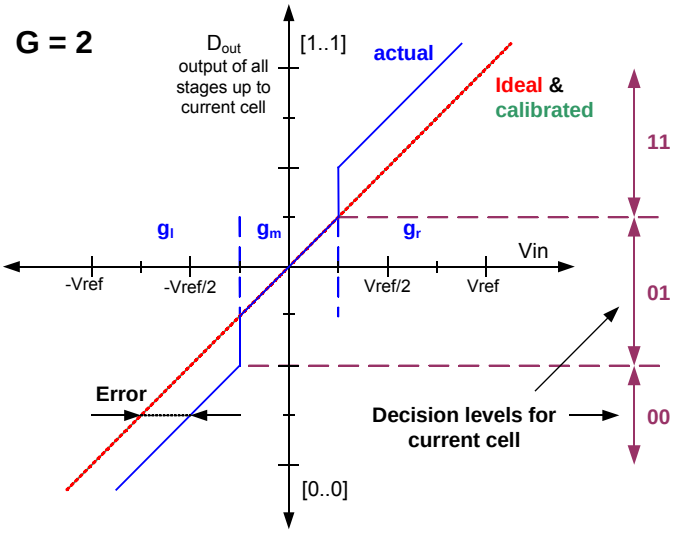


Figure 18: Calibrated output D_{out} up to stage n-1 with G=2

Due to the described 1.5bit cell architecture comprising three possible states there are also three different offset value combinations for the current cell *n*. If the mid-range line g_m is

taken as a reference for the outer range lines g_l and g_r the overall actual line can be expressed as follows:

$$D_{out}(n)=\begin{Bmatrix}\langle 11\rangle+D_{offset}\\ \langle 01\rangle\\ \langle 00\rangle-D_{offset}\end{Bmatrix}+\sum_{k=1}^{n-1}D_{out}(k)\;\equiv\;\begin{Bmatrix}g_l\\ g_m\\ g_r\end{Bmatrix}\;\;for\;\;\begin{matrix}V_{in}\geq V_{ref}/4\\ -V_{ref}/4<V_{in}<V_{ref}/4\\ V_{in}\leq V_{ref}/4\end{matrix}$$

with $\sum_{k=1}^{n-1}D_{out}(k)$ as the digitally corrected output from stage 1 to n-1

Before applying Digital Calibration there's a gap between the actual (blue) and the ideal (red) line and hence an error in the lower and upper range caused by comparator offset (g_l and g_r → $V_{ref}/4>V_{in}>V_{ref}/4$). Since only the upper cells are calibratable and the number of cells taken into account augments while moving up to the MSB cell (*see chapter 5.2.2*), the length of vector D_{out} (in #bits) varies from *n=#uncalibratable cells*+1bit to *n*=13bit according to the cell under evaluation.
To remove this error, Digital Calibration has to be applied: Depending on the state of cell *n* ($D_{out}(n)$=00, 01 or 11) either a constant is added ($V_{in}<-V_{ref}/4$) or subtracted ($V_{in}>V_{ref}/4$) for 00 and 11 respectively. Subsequently, as indicated by the green (calibrated) line the ideal and the actual line match.

As long as the inter-stage gain is steady the calibration process works fine. However, with gain G < 2 the following problem occurs:

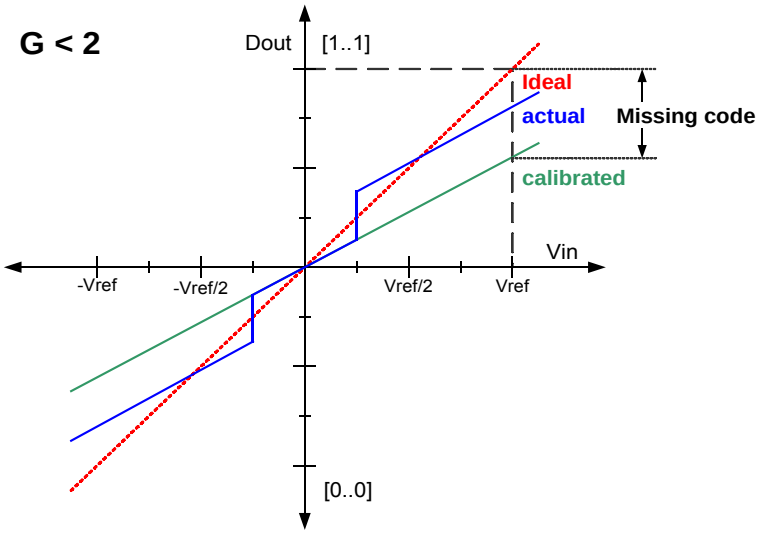


Figure 19: Calibration with G<2 resulting in Missing Code

If gain drops below G = 2 the calibrated line indicates that the stage output might not reach full scale [1..1] any more, which means that a code is skipped and therefore missing. So negative gain variations (G < 2) are absolutely fatal to the design and have to be prevented in any case. This is achieved by choosing a gain slightly above two, so it can never drop below.

Finally, there's the scenario in normal operation with gain slightly above two:
Now all codes are reached, there's just a small loss of resolution because the input voltage range is reduced and the maximum output D_{out} is reached before $V_{in}=V_{ref}$. Since this effect is comparably small the associated gain variation has to be exaggerated to make it visible.
The following figure illustrates this:

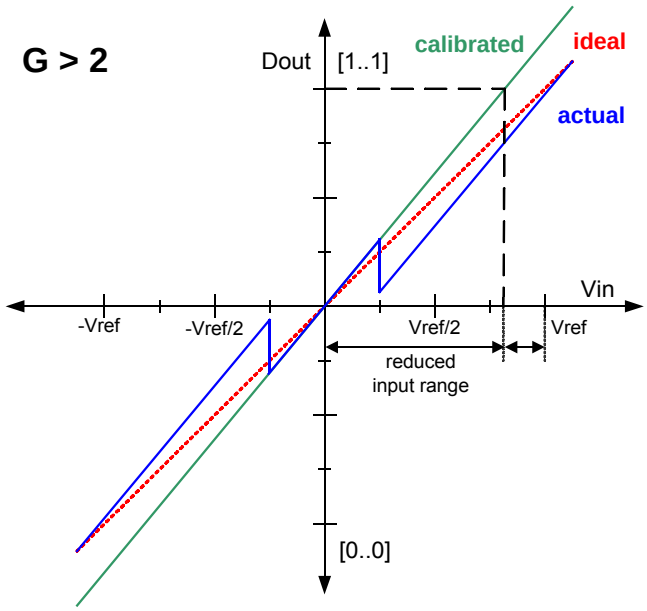


Figure 20: Calibration with G>2, normal operation

5.2.2 Calibration process of CRIC-II

CRIC has eight calibratable and five non-calibratable cells:

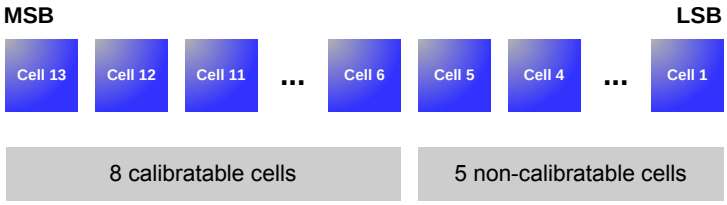


Figure 21: Calibratable cells

To correct the ADC output as shown in the previous chapter, two calibration constants have to be determined for each cell prior to the regular ADC operation. So there are the following steps to calculate these constants:

- First, four measurements given by the appropriate points in the 1.5bit cell transfer function have to be obtained for each cell (see *figure 22*).
- Second, the differences of each two vertically adjacent points have to be calculated.
- And finally the gathered constants have to be stored to make them available for later correction during all conversion cycles (the actual calibration).

Figure 22 illustrates the measurement process:

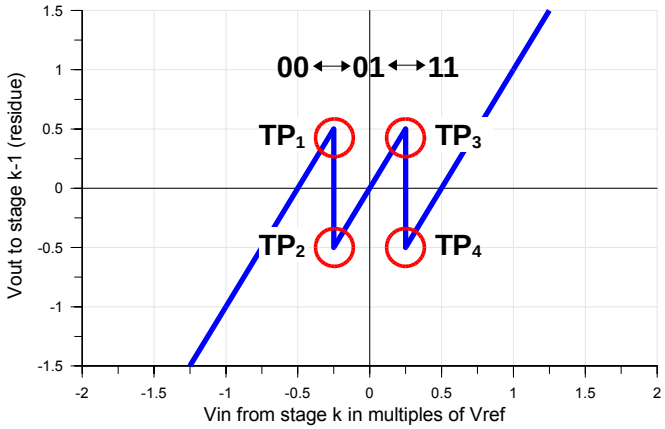


Figure 22: Transfer points to measure

To measure all indicted points of the transfer function each cell has the capability to be controlled from outside by a control register (and hence by the calibration logic). So there are several switches that represent the states each cell can have during operation. By using these switches the cells can be forced to the indicated points TP_1..TP_4.

Bit#	Function	Bit#	Function	Bit#	Function	Bit#	Function
1	Off8	16	CalMSB6	31	Off3	46	CalMSB1
2	CalVrefPlus8	17	CalLSB6	32	CalVrefPlus3	47	CalLSB1
3	CalVrefMinus8	18	Cal6	33	CalVrefMinus3	48	Cal1
4	CalMSB8	19	Off5	34	CalMSB3	49	Vpctrl drivers
5	CalLSB8	20	CalVrefPlus5	35	CalLSB3	50	CompsetM2 Ch4
6	Cal8	21	CalVrefMinus5	36	Cal3	51	CompsetM1 Ch4
7	Off7	22	CalMSB5	37	Off2	52	CompsetM2 Ch3
8	CalVrefPlus7	23	CalLSB5	38	CalVrefPlus2	53	CompsetM1 Ch3
9	CalVrefMinus7	24	Cal5	39	CalVrefMinus2	54	CompsetM2 Ch2
10	CalMSB7	25	Off4	40	CalMSB2	55	CompsetM1 Ch2
11	CalLSB7	26	CalVrefPlus4	41	CalLSB2	56	CompsetM2 Ch1
12	Cal7	27	CalVrefMinus4	42	Cal2	57	CompsetM1 Ch1
13	Off6	28	CalMSB4	43	Off1	58	Vpctrl Input amp
14	CalVrefPlus6	29	CalLSB4	44	CalVrefPlus1		
15	CalVrefMinus6	30	Cal4	45	CalVrefMinus1		

Table 3: CRIC-II configuration register

The table above illustrates all settings of CRIC's global control register. The yellow marked section stands for one group of adjustments that have to be controlled when taking measurements for calculating the calibration constants.

For example, a measurement cycle for cell#7 is established with the following settings:

	Off7	CalVrefPlus7	CalVrefMinus7	CalMSB7	CalLSB7	Cal7
1st measurement (TP_1)	0	1	1	0	0	1
2nd measurement (TP_2)	0	0	1	0	1	1
3rd measurement (TP_3)	0	1	0	0	1	1
4th measurement (TP_4)	0	1	0	1	1	1

Table 4: Measurement sequence

The entire calculation procedure comprises the following steps:

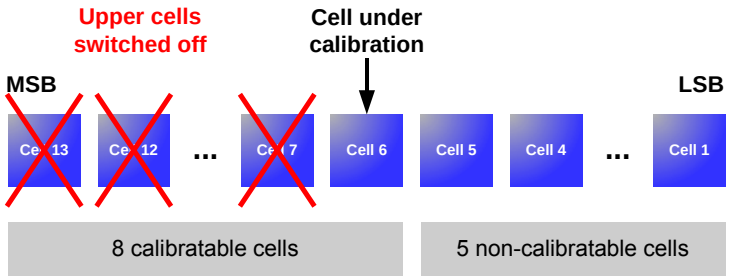


Figure 23: Calibration 1st step

- Start with cell#6
- All (MSB-) cells above are switched off (cell n_c to cell#13).
- Force cell under calibration n_c into four states as described in *table4*
- Acquire many measurements per point (~1000) and calculate average to reduce white noise
- After measuring TP_1 to TP_4 two constants can be calculated:
  $$\Delta_1(\text{cell } n_c) = TP_1 - TP_2 \quad \text{and} \quad \Delta_2(\text{cell } n_c) = TP_3 - TP_4$$
- Measure next cell: $n_c = n_c + 1$

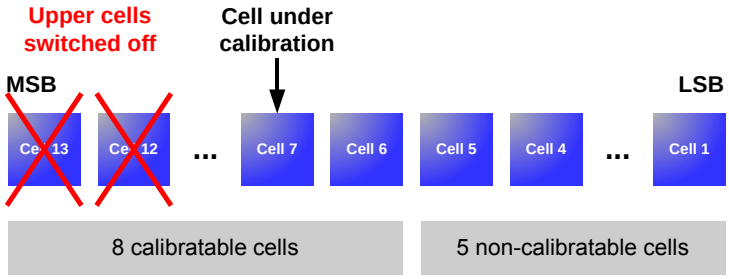


Figure 24: Calibration 2nd step

After repeating this sequence eight times, 8×2= 16 constants are calculated and have to be stored. The calibration is applied during any ADC acquisition cycle by following the decision table below:

Cell state binary	*Calibration*
00	Data - Δ_1
01	Do nothing
10	Data + Δ_2

Table 5: Calibration decision table

If a continuous and calibrated ADC acquisition is desired, the use of FPGA technology seems to be ideal because all mentioned corrections have to be accomplished in real-time. Furthermore there are fast FPGAs that support all necessary signal standards to control CRIC-II and to process all readout data. The implementation of the procedures particularly described in the previous chapter are explained in chapter 7.

6 Hardware setup

6.1 CRIC input and output signals

6.1.1 Pin assignment

As introduced in *chapter 1.3* the CRIC chip design embodies all blocks to readout a CCD, from input signal amplification to A/D conversion. To process four channels simultaneously the entire signal processing is implemented four times in parallel. Below you can see the CRIC pinout with the basic circuitry linked to these blocks:

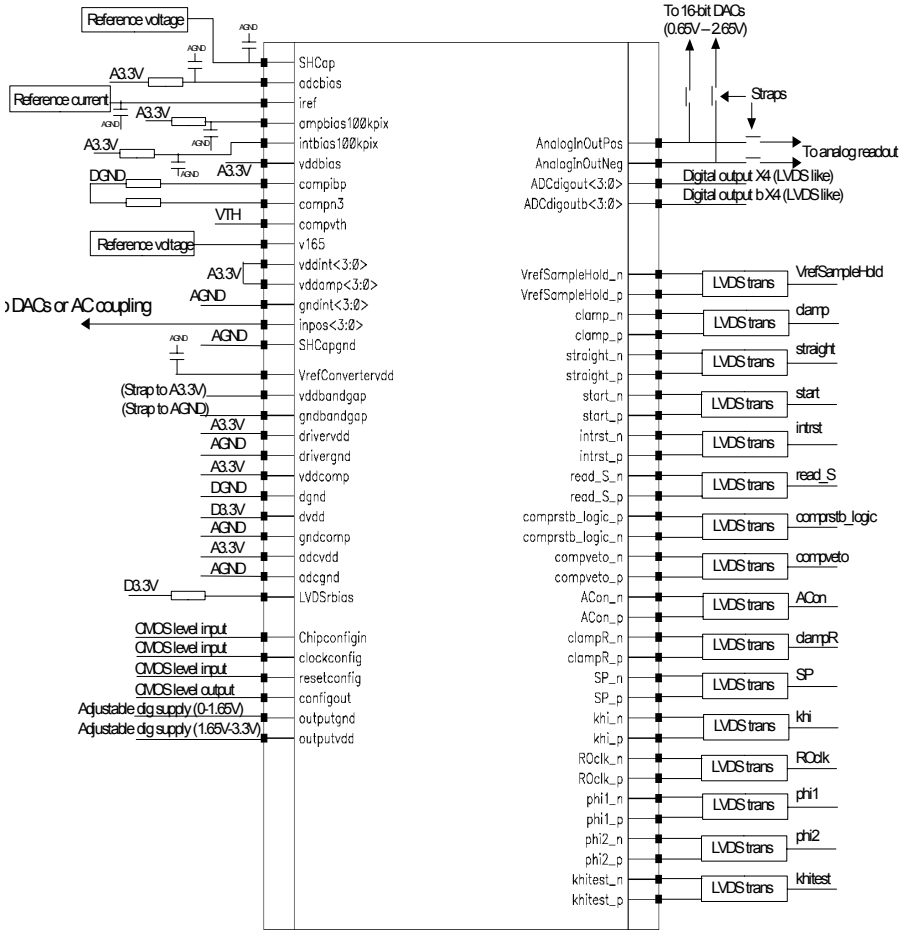


Figure 25: CRIC-II signals

The following table gives an overview of all CRIC-II signals with a short description. Please note, that some signals are only for chip design test purposes and don't have a particular functional background.

The most important signals are categorized in groups:
magenta=analog input, cyan=digital output, green=control pattern, yellow=configuration data

Pin#	Pin name	Description	Range
1	CONFIGOUT	Serial output of configuration register (test)	D 0 – 3.3V
2	GND	Substrate gnd	A 0V
3	COMPN3	Test integrator comp bias N3 (generated on chip)	D -10uA
4	COMPIBP	Comparator bias IBP (generated on chip)	D -10uA
5	VDDBIAS	Bias supply	A 3.3V
6	AMPBIAS100KPIX	Amplifier bias (generated on chip)	A 5uA
7	INTBIAS100KPIX	Integrator bias (generated on chip)	A 5uA
8	SHCAPGND	Bottom plate, GND on S&H cap on V165	A 0V
9	SHCAP	Top plate on S&H cap on V165 (generated on chip)	A 1.65V or a decoupling Cap
10	COMPVTH	Comparator threshold voltage	A 2.4V
11	COMPGND	Comparator gnd	D 0V
12	COMPVDD	Comparator supply	D 3.3V
13	ADCGND	ADC gnd	A 0V
14	ADCGND	ADC gnd	A 0V
15	ADCVDD	ADC supply	A 3.3V
16	ADCVDD	ADC supply	A 3.3V
17	ADCBIAS	ADC bias (generated on chip)	A 5uA
18	VrefConvertervdd	Vref Converter Vdd	A 3.3V
19	ANALOGINOUTNEG	Mulitslope negative output/ADC negative input	A 0.65V – 2.65V
20	ANALOGINOUTPOS	Mulitslope positive output/ADC positive input	A 0.65V – 2.65V
21	GND	Substrate gnd	A 0V
22	V165	Vref from Band gap reference	A 1.65V or 47nF Cap
23	IREF	Iref from Band gap reference	A 15uA or 47nF Cap
24	BANDGAPVDD	Band gap Vdd	A 3.3V
25	BANDGAPGND	Band gap gnd	A 0V
26	OUTPUTVDD	Digital output driver vdd	D 1.65V - 3.3V
27	OUTPUTVDD	Digital output driver vdd	D 1.65V - 3.3V
28	ADCDIGOUTB<3>	ADC channel 3 serial negative output	D 0V – 3.3V
29	ADCDIGOUT<3>	ADC channel 3 serial positive output	D 0V – 3.3V
30	ADCDIGOUTB<2>	ADC channel 2 serial negative output	D 0V – 3.3V
31	ADCDIGOUT<2>	ADC channel 2 serial positive output	D 0V – 3.3V
32	ADCDIGOUTB<1>	ADC channel 1 serial negative output	D 0V – 3.3V
33	ADCDIGOUT<1>	ADC channel 1 serial positive output	D 0V – 3.3V
34	ADCDIGOUTB<0>	ADC channel 0 serial negative output	D 0V – 3.3V
35	ADCDIGOUT<0>	ADC channel 0 serial positive output	D 0V – 3.3V
36	OUTPUTGND	Digital output driver gnd	D 0V – 1.65V
37	GND	Substrate gnd	A 0V
38	DGND	Digital gnd	D 0V
39	CLOCKCONFIG	Configuration register clock input	D 0 – 3.3V
40	CHIPCONFIGIN	Configuration input	D 0 – 3.3V
41	RESETCONFIG	Digital reset input	D 0 – 3.3V
42	GND	Substrate gnd	A 0V
43	VREFSAMPLEHOLD_P	Vref sample & hold LVDS positive input	D 1.2V ± 175mV
44	VREFSAMPLEHOLD_N	Vref sample & hold LVDS negative input	D 1.2V ± 175mV
45	SP_P	ADC output register load LVDS positive input	D 1.2V ± 175mV
46	SP_N	ADC output register load LVDS negative input	D 1.2V ± 175mV

Pin#	Pin name	Description	Range
47	ROCLK_P	ADC readout clock LVDS positive input	D 1.2V ± 175mV
48	ROCLK_N	ADC readout clock LVDS negative input	D 1.2V ± 175mV
49	DVDD	Digital Vdd	D 3.3V
50	DVDD	Digital Vdd	D 0 – 3.3V
51	KHI_N	Khi LVDS negative input	D 1.2V ± 175mV
52	KHI_P	Khi LVDS positive input	D 1.2V ± 175mV
53	PHI1_N	Phi1 LVDS negative input	D 1.2V ± 175mV
54	PHI1_P	Phi1 LVDS positive input	D 1.2V ± 175mV
55	PHI2_N	Phi2 LVDS negative input	D 1.2V ± 175mV
56	PHI2_P	Phi2 LVDS positive input	D 1.2V ± 175mV
57	READ_S_N	Slope latch LVDS negative input	D 1.2V ± 175mV
58	READ_S_P	Slope latch LVDS positive input	D 1.2V ± 175mV
59	COMPVETO_N	Comp veto LVDS negative input	D 1.2V ± 175mV
60	COMPVETO_P	Comp veto LVDS positive input	D 1.2V ± 175mV
61	COMPRSTB_LOGIC_N	Comp resetb LVDS negative input	D 1.2V ± 175mV
62	COMPRSTB_LOGIC_P	Comp resetb LVDS positive input	D 1.2V ± 175mV
63	KHITEST_N	Khitest LVDS negative input	D 1.2V ± 175mV
64	KHITEST_P	Khitest LVDS positive input	D 1.2V ± 175mV
65	GND	Substrate gnd	A 0V
66	ADCGND	ADC gnd	A 0V
67	ADCGND	ADC gnd	A 0V
68	ADCVDD	ADC supply	A 3.3V
69	ADCVDD	ADC supply	A 3.3V
70	INTRST_N	Integrator reset LVDS negative input	D 1.2V ± 175mV
71	INTRST_P	Integrator reset LVDS positive input	D 1.2V ± 175mV
72	START_N	Start/Stop LVDS negative input	D 1.2V ± 175mV
73	START_P	Start/Stop LVDS positive input	D 1.2V ± 175mV
74	STRAIGHT_N	CDS LVDS negative input	D 1.2V ± 175mV
75	STRAIGHT_P	CDS LVDS positive input	D 1.2V ± 175mV
76	CLAMPR_N	ClampR LVDS negative input	D 1.2V ± 175mV
77	CLAMPR_P	ClampR LVDS positive input	D 1.2V ± 175mV
78	CLAMP_N	Clamp LVDS negative input	D 1.2V ± 175mV
79	CLAMP_P	Clamp LVDS positive input	D 1.2V ± 175mV
80	ACON_N	Clamp LVDS positive input	D 1.2V ± 175mV
81	ACON_P	Clamp LVDS negative input	D 1.2V ± 175mV
82	LVDSRBIAS	LVDS receiver bias	D 250uA
83	DGND	Digital gnd	D 0V
84	GND	Substrate gnd	A 0V
85	GNDINT<0>	Ch0 Integrator gnd	A 0V
86	GNDAMP<0>	Substrate gnd	A 0V
87	INPOS<0>	Ch0 Amplifier input	A 1.65V – 2.15V
88	VDDAMP<0>	Ch0 Amplifiers VDD supply	A 3.3V
89	VDDINT<0>	Ch0 Integrator VDD supply	A 3.3V
90	GNDINT<1>	Ch1 Integrator gnd	A 0V
91	GNDAMP<1>	Substrate gnd	A 0V
92	INPOS<1>	Ch1 Amplifiers input	A 1.65V – 2.15V

Pin#	Pin name	Description	Range
93	VDDAMP<1>	Ch1 Amplifiers VDD supply	A 3.3V
94	VDDINT<1>	Ch1 Integrator VDD supply	A 3.3V
95	GNDINT<2>	Ch2 Integrator gnd	A 0V
96	GNDAMP<2>	Substrate gnd	A 0V
97	INPOS<2>	Ch2 Amplifier input	A 1.65V – 2.15V
98	VDDAMP<2>	Ch2 amplifiers VDD supply	A 3.3V
99	VDDINT<2>	Ch2 integrator VDD supply	A 3.3V
100	GNDINT<3>	Ch3 Integrator gnd	A 0V
101	GNDAMP<3>	Substrate gnd	A 0V
102	INPOS<3>	Ch3 Amplifier input	A 1.65V – 2.15V
103	VDDAMP<3>	Ch3 amplifiers VDD supply	A 3.3V
104	VDDINT<3>	Ch3 integrator VDD supply	A 3.3V
105	GND	Substrate gnd	A 0
106	DRIVERVDD	Driver vdd	A 3.3V
107	DRIVERGND	Driver gnd	A 0V
108	DRIVERGND	Driver gnd	A 0V
109	BACKSIDEGND	Substrate gnd	A 0

6.1.2 Signal standards

Since some of the signals operate in the frequency range of f= 1..40MHz and low jitter and especially low skew is very important for ADC accuracy, the use of a signal standard capable of supporting these requirements is imperative for the design. *Table 6* illustrates, that LVDS (Low Voltage Differential Signal) is ideal for this application because it is a low voltage, high speed and differential signal standard and seems to be superior to competing technologies. All other signals of CRIC-II that aren't timing critical comply to the LVTTL standard.

Advantages	LVDS	PECL	Optics	RS-422	GTL	TTL
Data rate up to 1Gbps	+	+	+	-	-	-
Very low skew	+	+	+	-	+	-
Low dynamic power	+	-	+	-	-	-
Cost effective	+	-	-	+	+	+
Low noise/EMI	+	+	+	-	-	-
Single power supply/reference	+	-	+	+	-	+
Migration path to low voltage	+	-	+	-	+	+
Simple termination	+	-	-	+	-	+
Wide common-mode range	-	+	+	+	-	-
Process independent	+	-	+	+	+	+
Allows integration w/digital	+	-	-	-	+	+
Cable breakage/splicing issues	+	+	-	+	+	+
Long distance transmission	-	+	+	+	-	-
Industrial temp/voltage range	+	+	+	+	+	+

Table 6: Comparison of signal standards

6.2 CRIC-II test board

Figure26 shows the CRIC-II test board with all connections to the surrounding test equipment, which is in first line a PC running special test software to acquire measurements and to control the onboard test DACs and the FPGA. The FPGA board is connected as a daughter board via two high density connector.

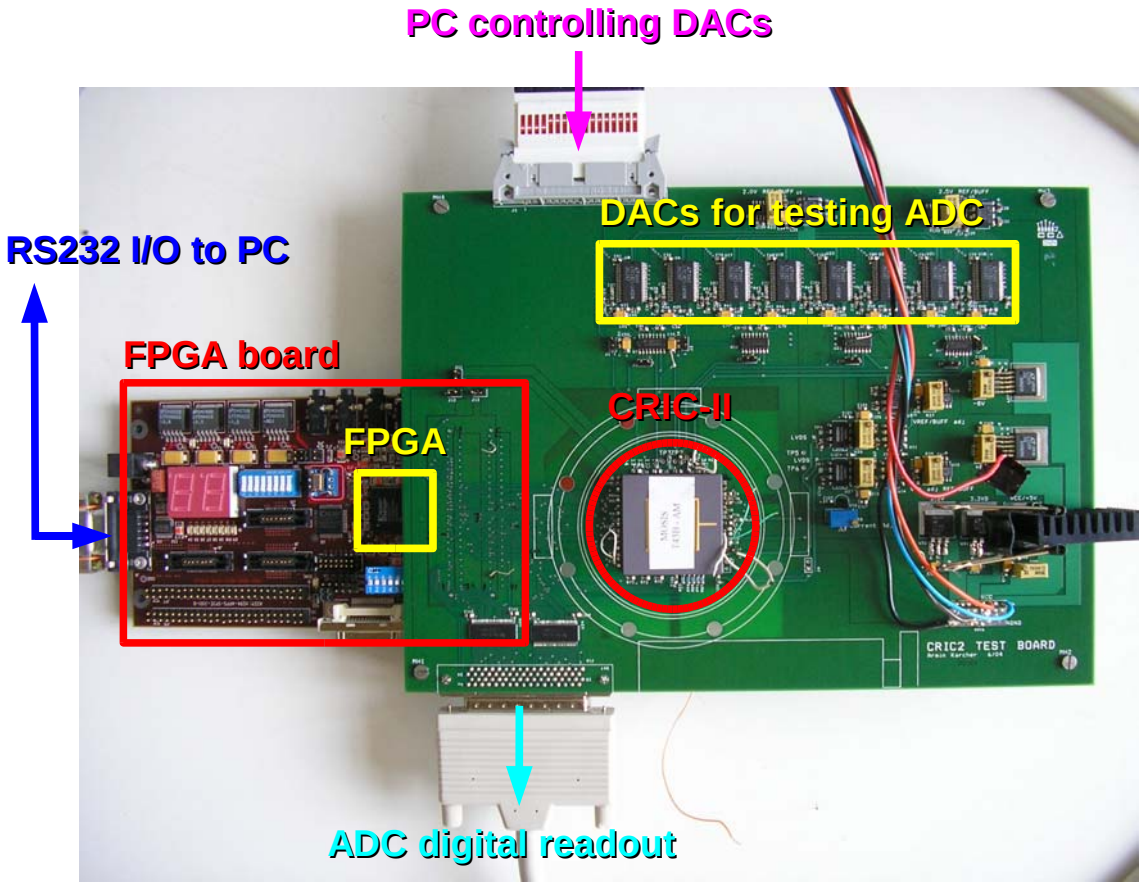


Figure 26: CRIC-II test board setup

The two connectors on the top and the bottom of the board for controlling the DACs and to acquire ADC readout data are connected to a PCI digital I/O card by National Instruments. This card provides readout data to the measurement software and grants access to the onboard DACs that are used to generate signal pattern for all channels of CRIC-II.

6.3 FPGA board

The requirements to operate CRIC-II include:

- Test pattern generation, i.e. driving of all acquisition related input signals (LVDS!)
- Real-time control of CRIC via RS-232 connection
- Receiving of measurement related data via RS-232
- Acquisition and deserializing of readout data
- Digital Correction
- Digital Calibration

To fulfill all listed requirements something really versatile has to be considered. Microcontrollers drop out because they don't support a fixed timing scheme and only the faster ones provide the required speed. So programmable logic like FPGAs seem to be the best solution.

The Spartan-II FPGA series by XILINX offers a very good performance and capacity for a reasonable price. Furthermore the synthesis software is free for use with FPGA ≤ 200 kGates. However, when considering larger FPGAs the cost of placement has to be considered. Mostly, theses FPGAs have a BGA (Ball Grid Array) package, which makes the placement for a smaller lot size more expensive than the silicon itself. Fortunately there are several manufacturers who offer FPGA boards with numerous functions. These boards are designed in a way that they aren't only usable for evaluation of the FPGA but to be included as a part of the design, e.g. as a daughterboard. For this reason a board from Avnet was used. This board is with a price of about $250 comparably inexpensive and offers all desired functions:

- Xilinx Spartan or Virtex FPGA with about 200kGates
- LVDS support for at least 38 pairs
- Various connectors
- Onboard PROM (flash type)
- Support of fast (parallel) PC synthesis programming cable
- Display for debugging (LEDs or display)
- Exchangeable crystal oscillator to adjust operation frequency

Figure 27: Avnet Spartan-IIE kit

As the image might reveal, the selected board provides even a lot more features than essentially wanted. However all wanted features are implemented including a Spartan-IIE (E=enhanced, due to LVDS support), many connectors and some displays and switches. The board even embodies a SUB-D9 connector and hence the possibility to synthesize a UART connection to a PC.

7 FPGA design implementation

7.1 Design overview

Figure28 illustrates the connections between all blocks of the design:

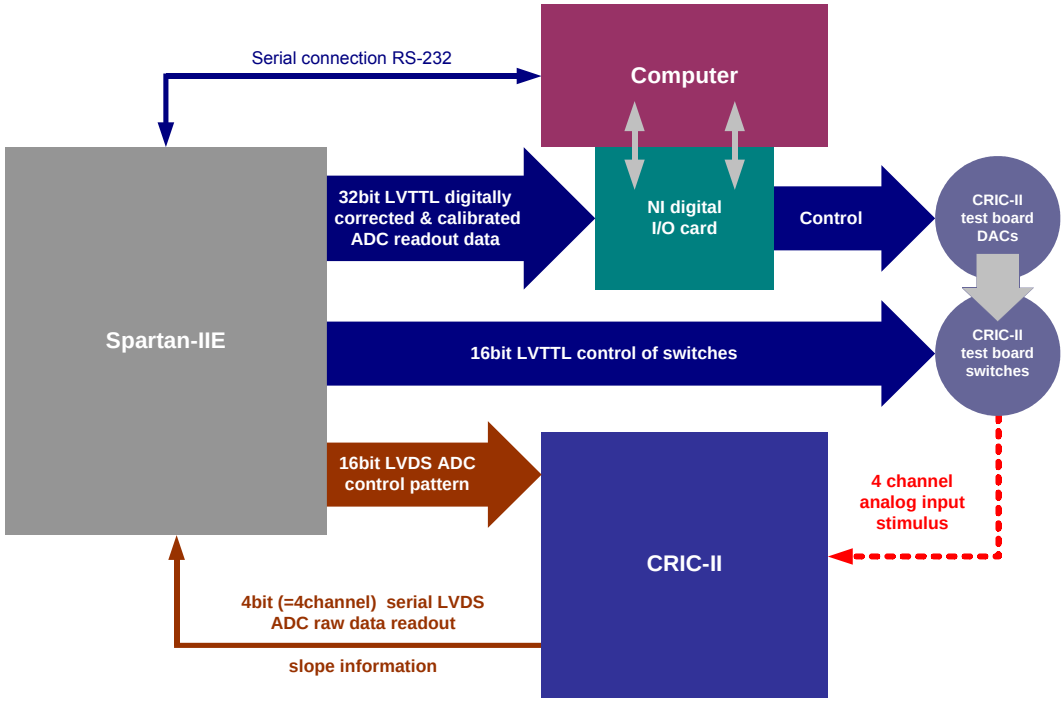


Figure 28: CRIC-II data flow

The entire FPGA design is programmed using VHDL
(VHSIC Hardware Description Language, VHSIC=Very High Speed Integrated Circuit).

7.2 Timing diagrams

To control CRIC-II during operation the signal timing according to the following diagrams have to be implemented (see *chapter 6.1* for a description of the signals). This pattern generation is crucial to control the behavior of CRIC both during normal conversion and calibration. Since the pattern is stored in multiple bit vectors and is accessible from outside via serial connection (RS-232) it allows a flexible handling and adjustment of the ADC timing throughout testing and customizing.

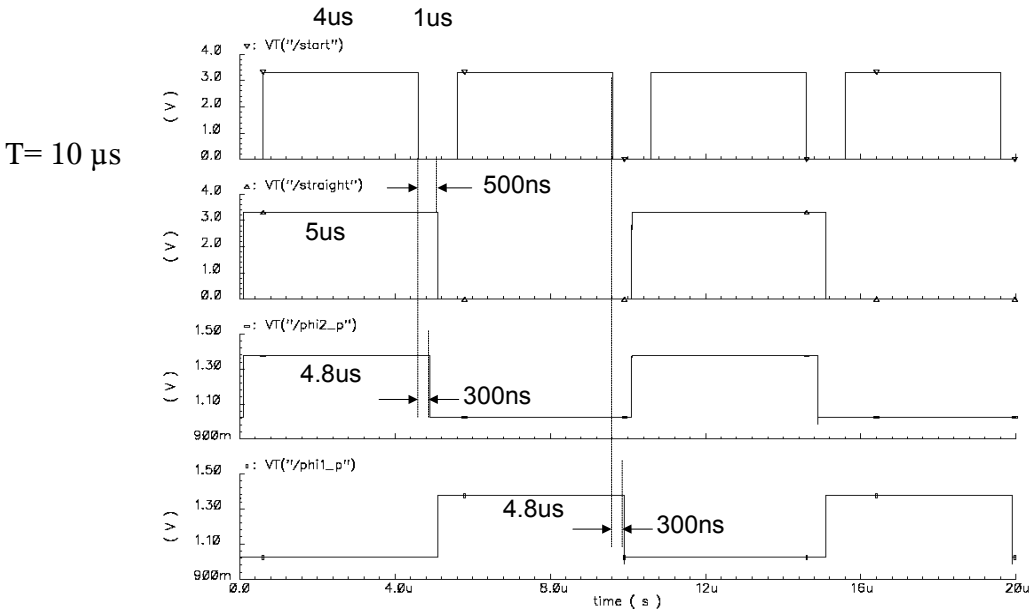


Figure 29: CRIC-II timing diagram: Start/stop of conversion cycle

A complete conversion of all input channels is finished after 10 µs.

The next diagram shows the relation between readout clock cycles and the number of cycles that have to be run to finish a complete measurement readout.

After 28 readout clock cycles a complete measurement is acquired.
→ 28bit = 26bit raw ADC data + 2bit slope information.

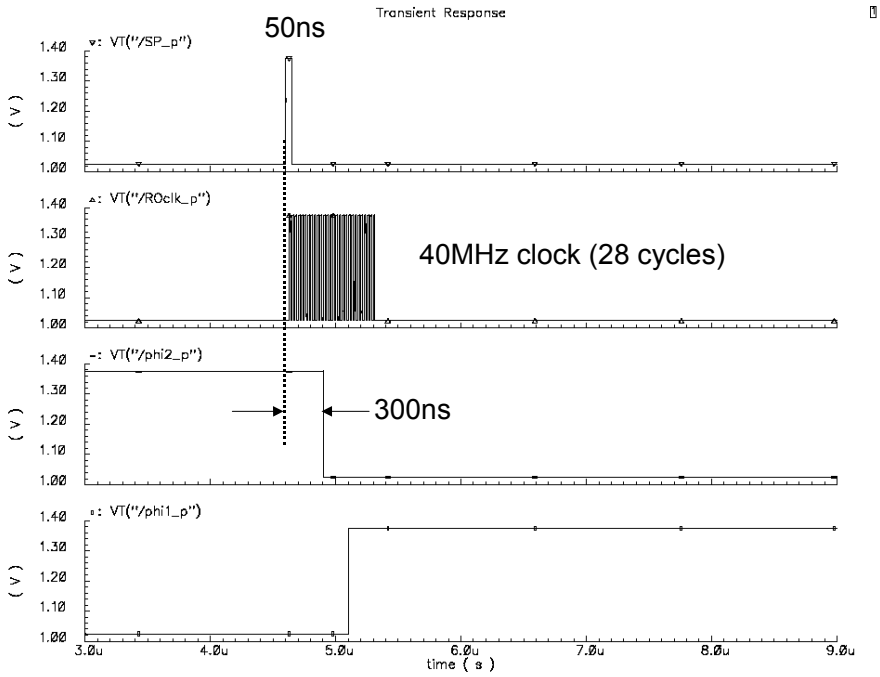


Figure 30CRIC-II timing diagram: Readout clock and data acquisition

To show the signal transitions precisely, the indicated area has to be zoomed:

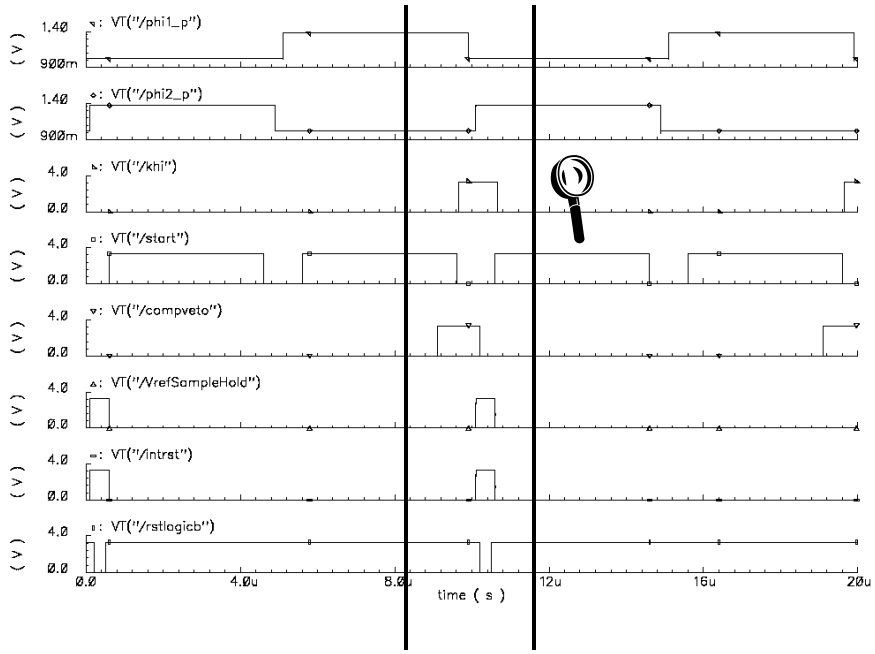


Figure 31: CRIC-II timing diagram: Zoom area

Zoom into *figure31* to clarify the delay between the signals:

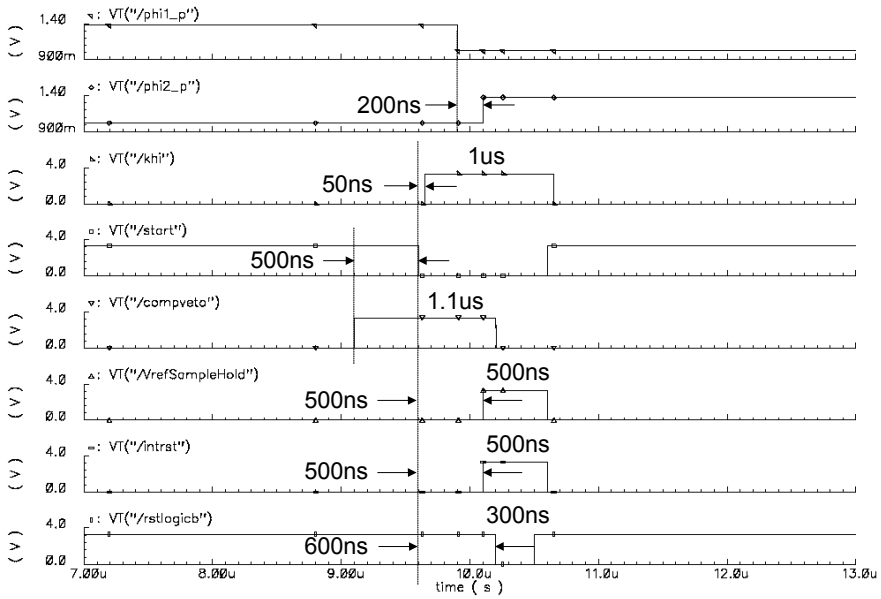


Figure 32: CRIC-II timing diagram: Delay between signals

The illustrated timing diagrams are encoded and stored into the FPGA block-RAM to drive the signals in realtime and accordingly to control CRIC-II during operation. The calculations related to memory usage and clock frequency are presented in the following chapter. An in-depth description of the signal timing can't be given here, because it would need a complete explanation of the ADC at circuit level which goes beyond the scope of this thesis.

Nevertheless, to give an idea of the function of the timing pattern take a glance at the next figure. For instance it shows *phi1* and *phi2* and the related switches that have to be controlled while passing data through the 1.5bit cells of the ADC.

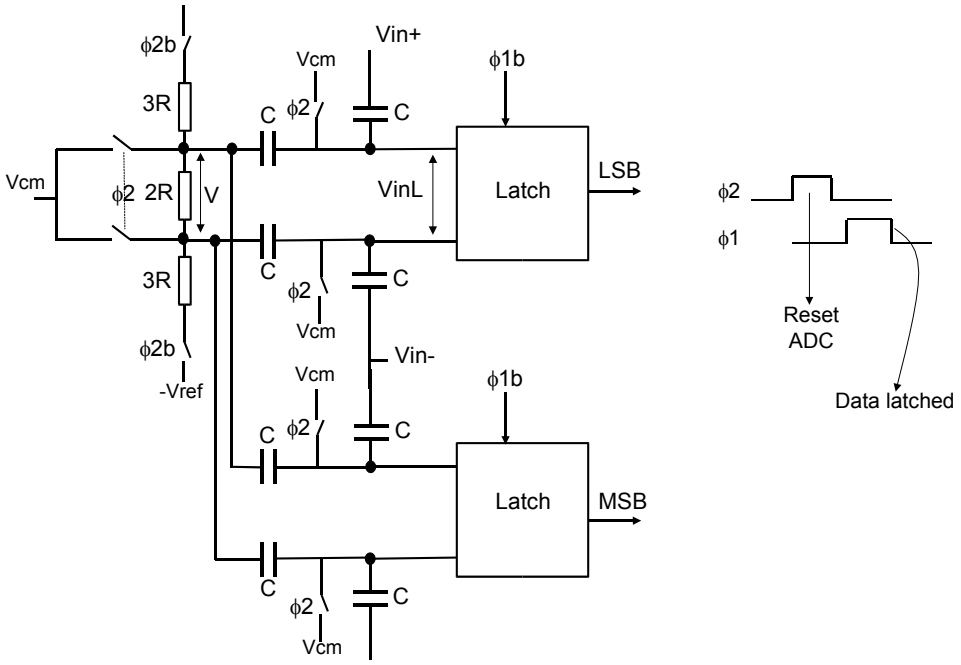


Figure 33: 1.5bit cell schematic and signal timing of phi1/2

7.3 Design considerations

The desired readout sampling rate of f_s=100kHz and a timing accuracy of T_{acc}=25ns (*see chapter 7.2*) leads to the following calculations:

Sampling period: $$T_{sample} = \frac{1}{f_s} = 10\,\mu s$$

For a pattern output with 1bit per cycle it needs a minimum clock frequency of:

$$f_{clock} = \frac{1}{T_{acc}} = 40\,MHz$$

Therefore the amount of memory necessary to hold one signal over a complete period is determined by:

$$n = \frac{T_{sample}}{T_{acc}} = 400 \quad \text{time slices}$$

7.3.1 Pattern generation:

Using the number of time slices, it is possible to calculate the maximum memory usage necessary to drive all blocks of CRIC-II (including four channels):

Number of signals to drive: 16 CRIC-II pattern (LVDS) + 16 switches (LVTTL) = 32

Number of bits/cycle: $$n = \frac{32\,signals}{1\,cycle} = 32\,bit/25\,ns$$

Max. memory usage: $$A_{mem} = (32\,bit/slice)\cdot 400\,slices = 12800\,bit = 12.8\,kbit$$

Since the Spartan-IIE architecture includes 14×4096bit block RAM with a variable data width from 1bit to 16bit, four Block-RAMs are necessary to have parallel access to 32bit per cycle.

7.3.2 Storage of calibration constants

As demonstrated each calibratable ADC cell is associated with two calibration constants. The memory usage for one set of calibration constants is:

$$A(cell[n=6..13]) = \sum_{n=5}^{12} n = 68\,bit$$

Memory usage for all calibration constants: 2×68bit = 136bit (minimal)

However, to have access to all constants in the same way, data length is kept constant, i.e. two block RAMs with a data width of 16bit per block are used to be able to read two adjacent calibration constant at the same time. Thus, the actual memory usage can be calculated:

$$A(cell[n=6..13]) = 8\,cells\cdot 16\,bit = 128\,bit$$

Memory usage of all calibration constants (of one channel): 2×128bit = 256bit

7.4 State machines

The FPGA implementation comprises five Finite State Machines (FSM) that are all of type Mealy. There are various additional processes that aren't FSM structured, but the crucial control functions are maintained by them (*see Appendix: Code documentation*).
The list below gives an overview:

- **SST: Main state machine (serial UART state machine):**
  Controls the UART communication, i.e. receives commands from serial terminal (9600,8N1), stores the received pattern into the block RAM, sends data and starts/stops other processes and state machines
- **PST: Pattern generation state machine:**
  Runs the pattern received and stored by SST. This FSM is controlled by SST and DST.
- **CST: Configuration state machine:**
  Sends 58bit of configuration data received by SST to CRIC-II. This FSM is controlled by SST and DST.
- **DST: Digital Calibration state machine:**
  Calculates the calibration constants and stores them into the block RAM. This FSM is controlled by SST.
- **FST: Final value state machine:**
  If calibration is switched on, this FSM calculates the calibrated ADC values based on the calibration constants. FST isn't controlled by any other state machine, it only triggers on new readout data.

7.5 Subcomponents

Basically subcomponents are portable and independent of higher layers in hierarchy, i.e. these code fragments can be instantiated in any other VHDL design. The implemented subcomponents are hierarchically subordinate, because they're all dependent on the main architecture:

- **ADCDigCorr:** Applies Digital Correction and masks out MSB bits during calculation of calibration constants (*see chapter 5.1*)
- **RAM256x16:** Synthesizes RAM consisting of FPGA block RAM cells with 16bit data and a cell depth of 256 cells
- **RAM512x16:** Synthesizes RAM consisting of FPGA block RAM cells with 16bit data and a cell depth of 512 cells
- **BinToHex:** Transcodes a 16bit binary input vector to an 8bit hexadecimal ASCII representation according to the position given by *pos* ranging from 3 down to 0 (0=LSB). This component is used to process the calculated calibration constants and to make them available via UART. Therefore this component is only required to control and to check the calculation during evaluation and debugging of the design.
- **MiniUART:** Provides all fundamentals to implement a standard UART. MiniUART by Philippe CARTON is a free IP core and complies to the Wishbone interface standard.

7.6 State charts

On the following pages there's a complete catalogue of state charts for all state machines and processes of the design. The charts are derived from the FPGA source code using a vector based diagramming software that supports drawing of UML (Unified Modeling Language) state charts. This type of state charts represent one of the most convenient and complete ways to describe a source code.

7.6.1 Overview

To get into the organization of the charts an overview is given below. Indicated by the blue boxes there are the five FSMs (*see chapter 7.4*). The red boxes represent the two output processes to deliver the final ADC output data and to generate the signal pattern to control CRIC-II. In general, a box stands for a potential state whereas an arrow placed between either represents a condition or an action associated with the state transition.

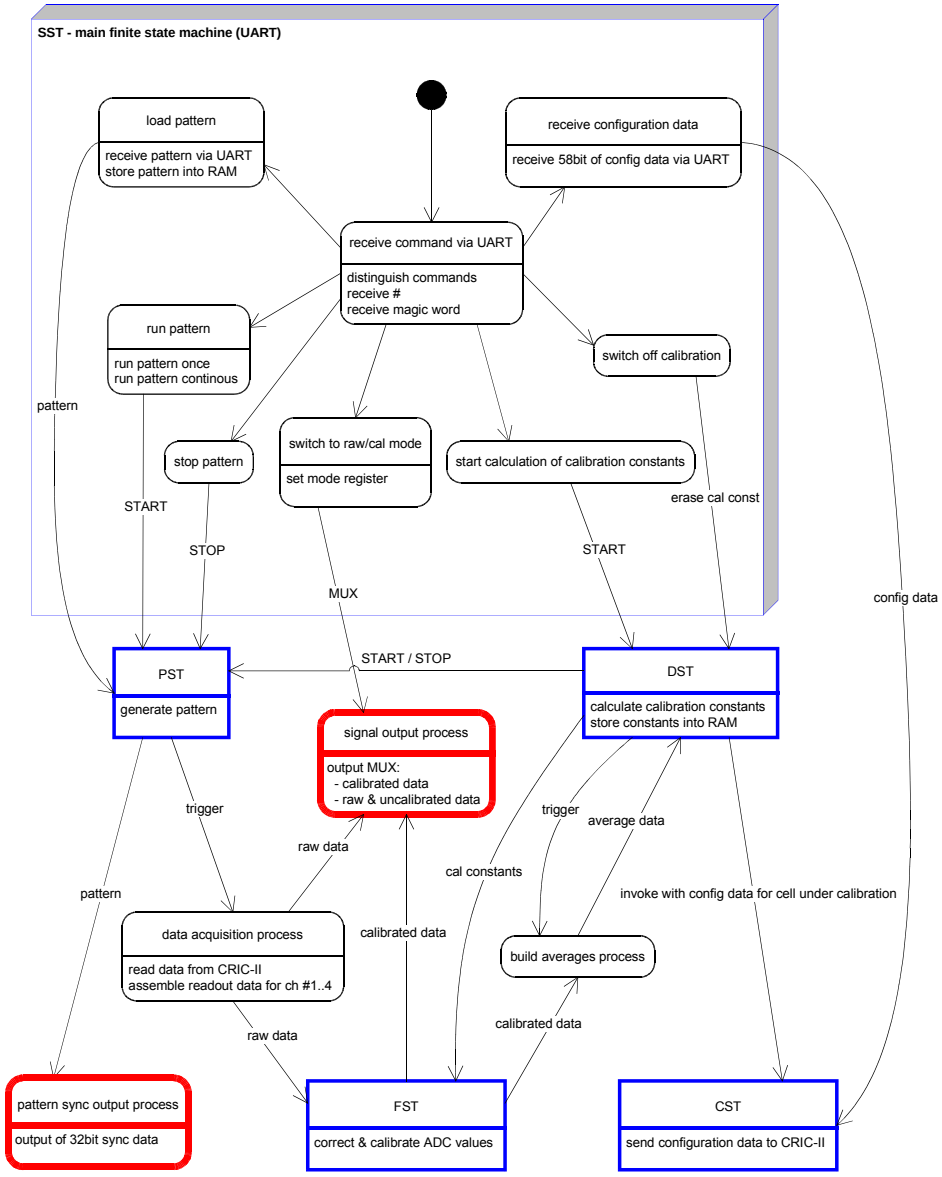


Figure 34: State chart overview of all FSMs

7.6.2 Main state machine (SST)

The main state machine having the highest hierarchical level in design is the serial state machine that receives all commands via UART from the computer and controls most of the other FSMs.

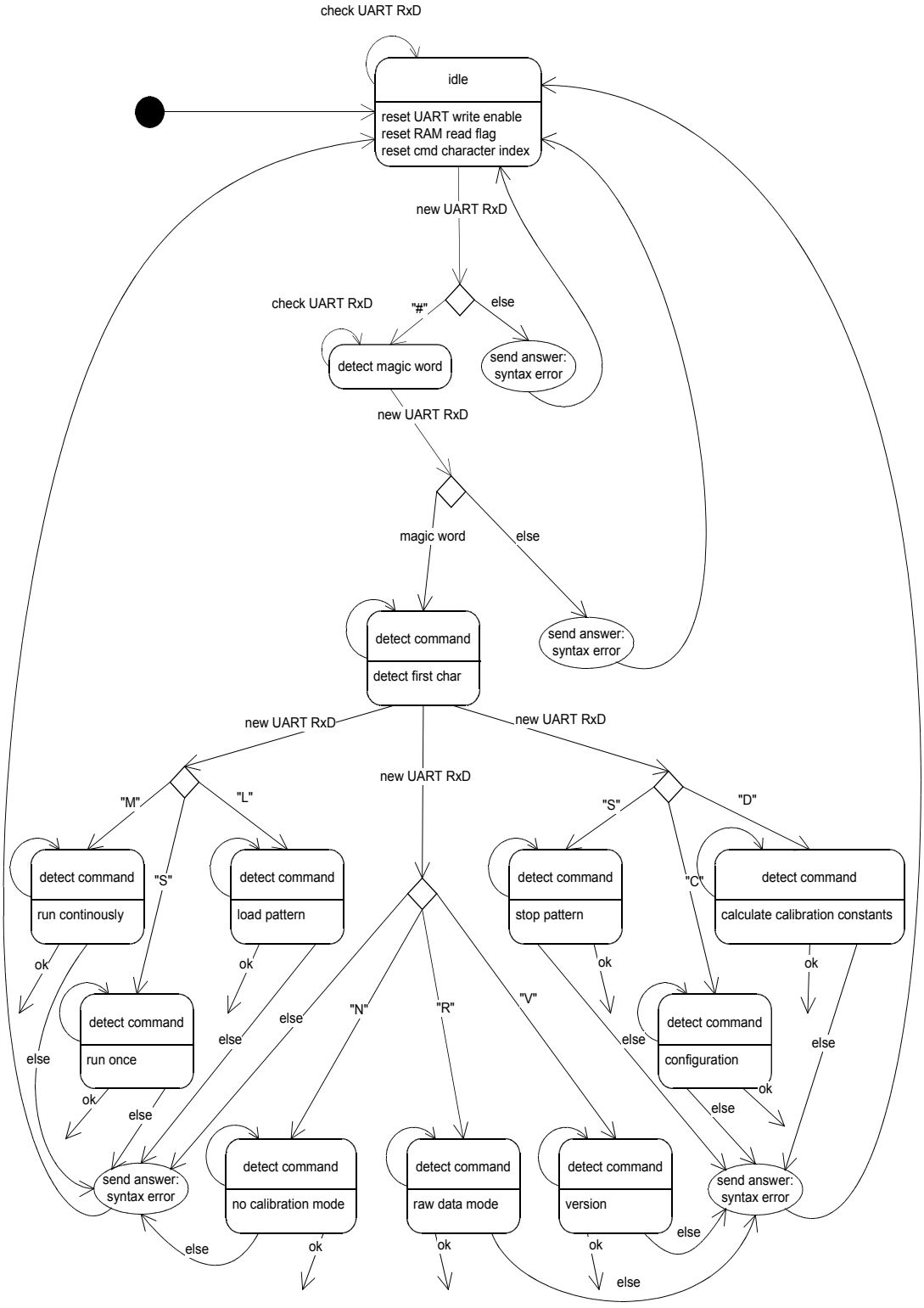


Figure 35: State chart SST(main) - Detect command

Figure 35 illustrates the SST command chain from top to bottom. Every command has to be initiated by a "#" symbol followed by a Magic word. Subsequently the actual command is expected. According to the chart branches detecting the first character of a command (=receiving of a new UART byte), a valid detection can be:

- *MRN:* Starts continuous acquisition
- *RNO:* Starts single acquisition
- *STP:* Stops acquisition
- *LDP:* Loads ADC control pattern via UART
- *CFG:* Loads CRIC-II configuration register via UART
- *VER:* Sends software version
- *DEC:* Starts calculation of a new set of calibration constants
- *NCL:* Switches off Digital Calibration and sends only digitally corrected data
- *RAW:* Switches off Digital Correction and sends raw ADC data

All commands are then either answered with an okay, an error (syntax or checksum), the request for more data (commands LDP and CFG) or by sending the calculated calibration constants (DEC). For more detail please see code documentation (*Appendix 10.3*).

Subroutine: Run Pattern (MRN/RNO)

The following charts are subroutines invoked after the appropriate command is detected (*see figure 35:* actions taking place right after the arrows labeled with ok):

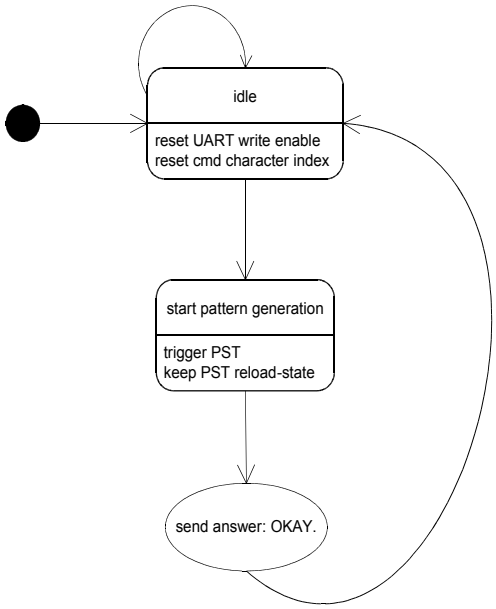


Figure 36: State chart SST - Run pattern continuously

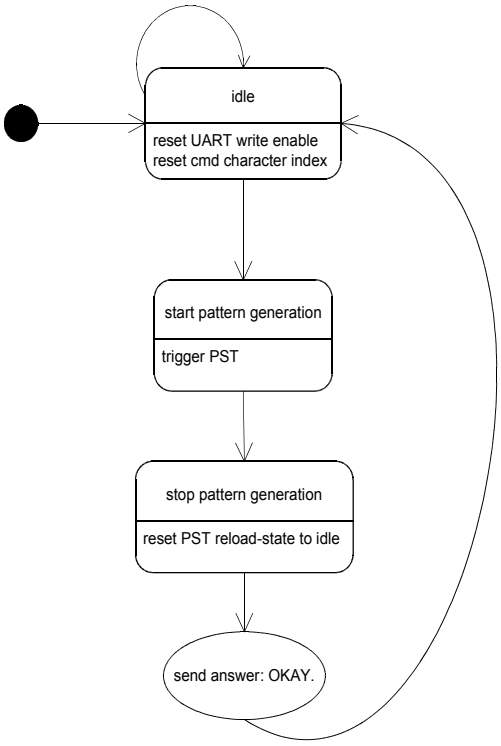


Figure 37: State chart SST - Run pattern once

Figure 36 and *figure 37* show how the pattern state machine (PST) is set to single or continuos run mode by either resetting or keeping PST's reload state register.

Subroutine: Load pattern (LDP)

Figure38 shows how the pattern is received via UART and stored into two 16bit block-RAMs. The length of the pattern is fixed: 12800bit=1600byte plus checksum (*see chapter 7.3.1*). The checksum is calculated as the modulo-8 sum of all incoming data bytes.

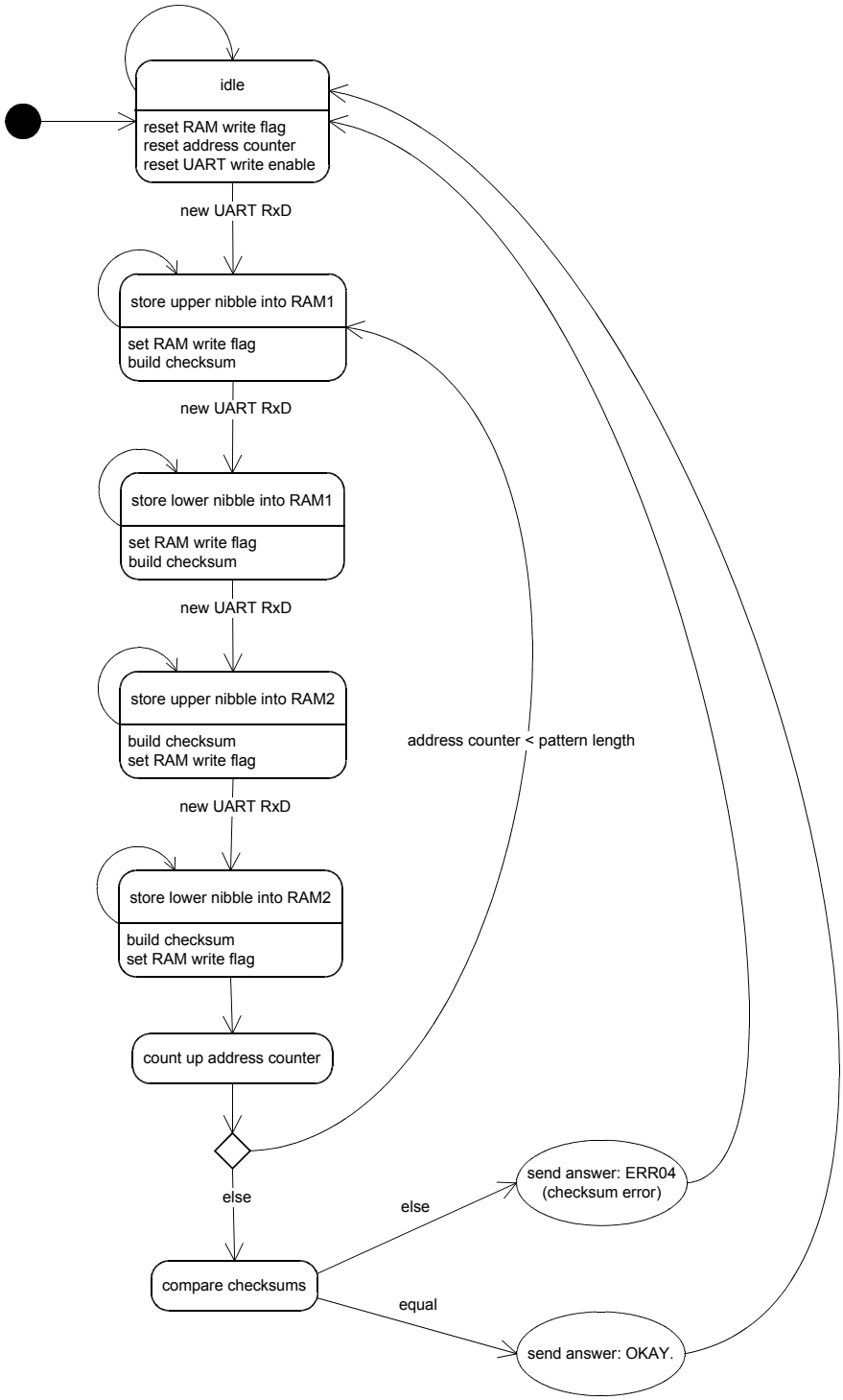


Figure 38: State chart SST - Load pattern

Subroutine: Stop pattern (STP)

The pattern generation (PST) is stopped by resetting the state reload register to idle, as indicated in *figure 39*.

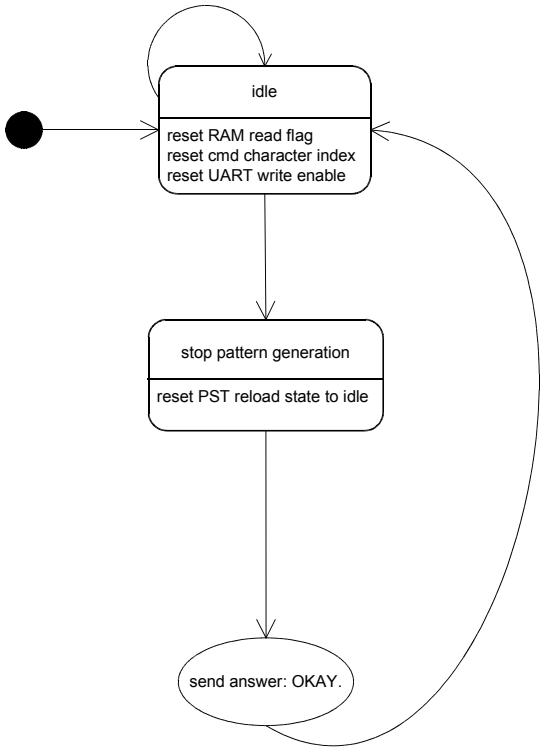


Figure 39: State chart SST - Stop pattern

Subroutine: Send software version (VER)

This command sends the software version currently running on the FPGA (*figure 40*).

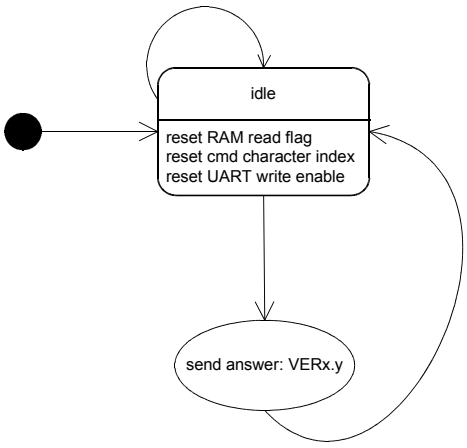


Figure 40: State chart SST - Send version

Subroutine: Load CRIC-II configuration register (CFG)

To load CRIC-II with different configuration sets (*see chapter 5.2.2*) it is possible to transmit data via UART to the FPGA and to forward it to CRIC afterwards. Since the configuration register has a length of 58bit the subroutine shown in *figure 41* receives 7×8bit + 2bit =58bit. The entire set is secured by a modulo-8 checksum. If the transmission is error-free the state machine responsible for sending configuration sets (CST) is triggered to output the received data to CRIC.

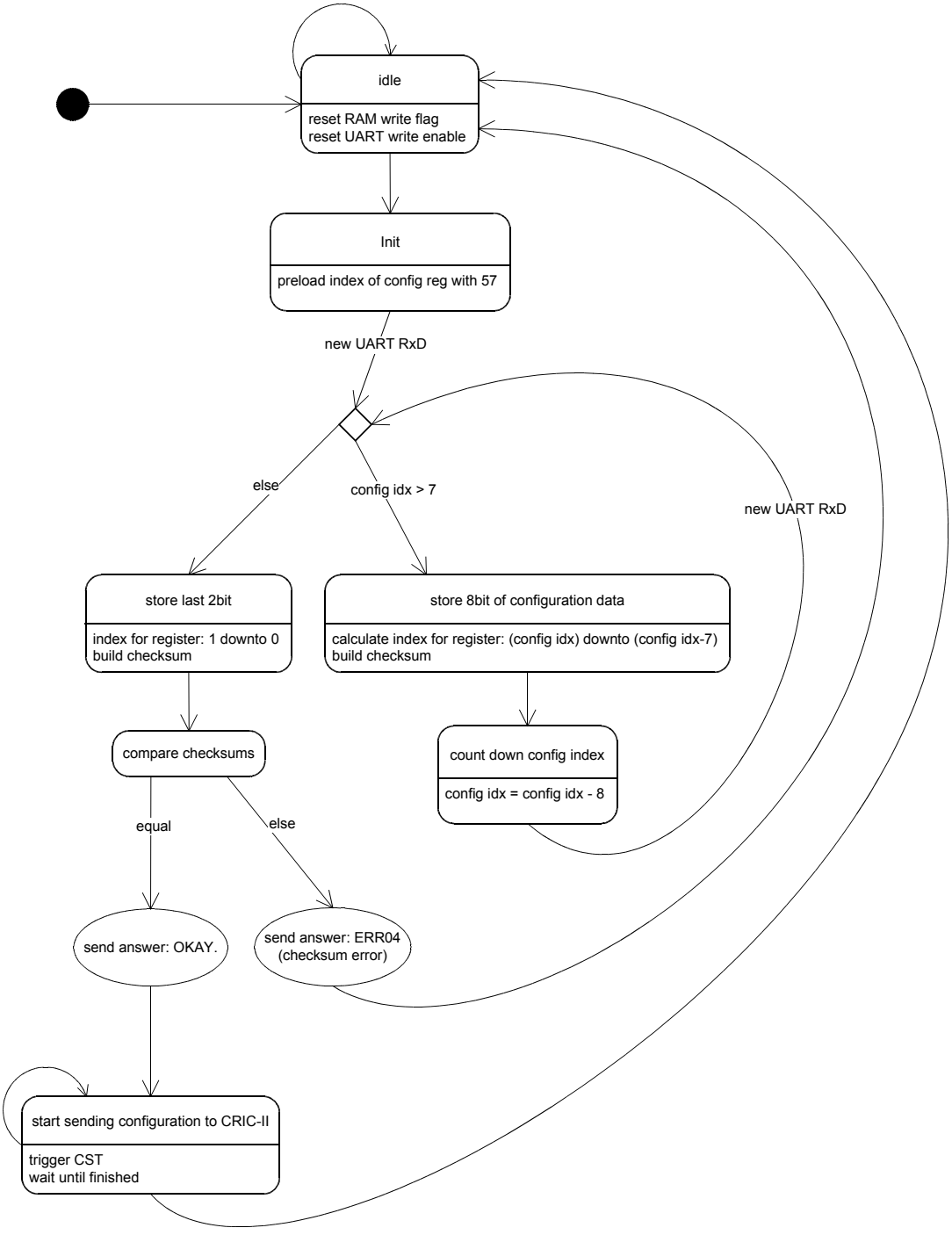


Figure 41: State chart SST - Record config data from UART

Subroutine: Start calculation of calibration constants (DEC)

Basically the subroutine shown in *figure 42* triggers the DEC state machine to calculate new calibration constants. This includes the following steps:

- Invoke DEC to erase related memory areas storing previous constants
- Invoke DEC to calculate a new set of constants
- Wait until calculation is done
- Convert calibration results using subcomponent BinToHex (*see chapter 7.5*)
- Send converted calibration constants via UART to external terminal (computer)

Please note, that the last two steps and hence the bigger part of this routine doesn't have a core-functional background. They're just used to evaluate the ADC.

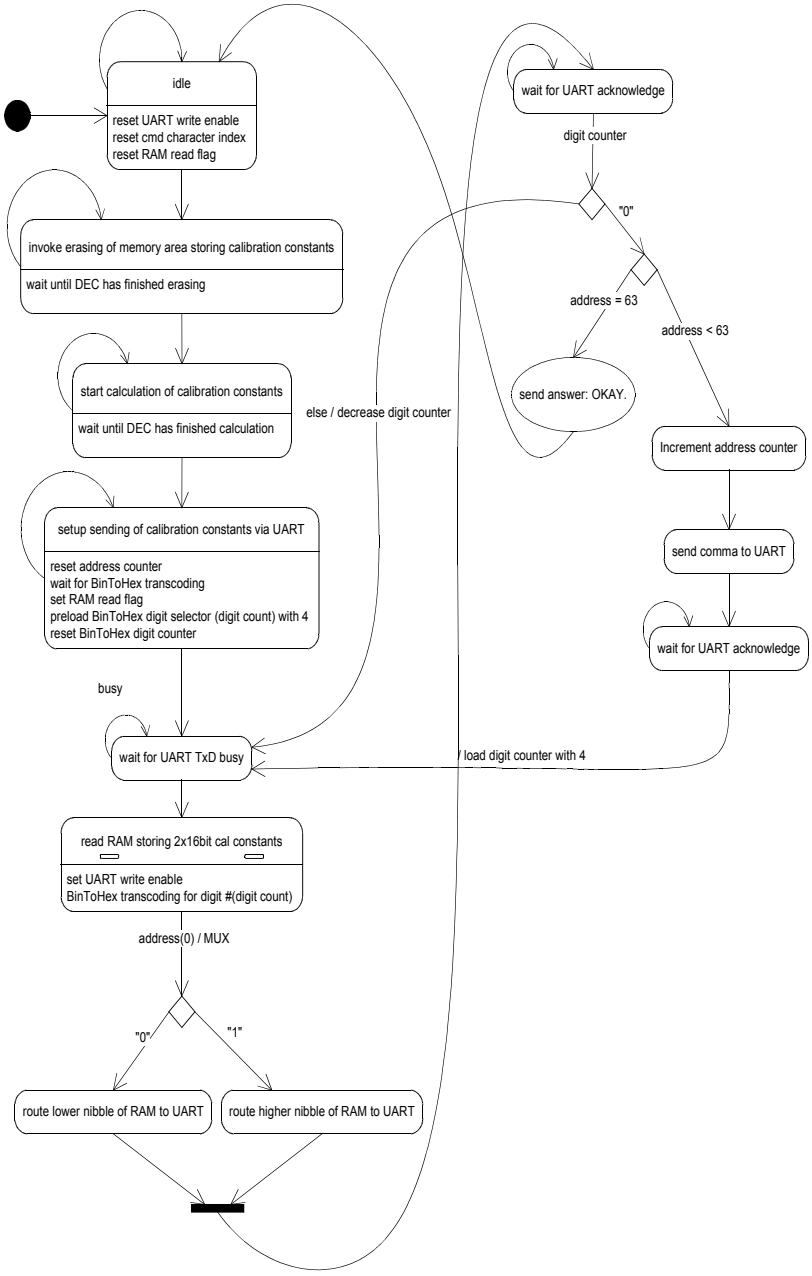


Figure 42: State chart SST - Start calculation of calibration constants

Subroutine: Switch to uncalibrated data output mode (NCL)

Figure 43 demonstrates how switching to uncalibrated data mode works. It's simply achieved by writing 0x00 to all constants (by invoking DST). This way the data flow is kept constant and the actual calibration state machine (FST) runs all the time, it just calibrates with zero correction. Please keep in mind that even if switched to uncalibrated mode the output data is still digitally corrected.

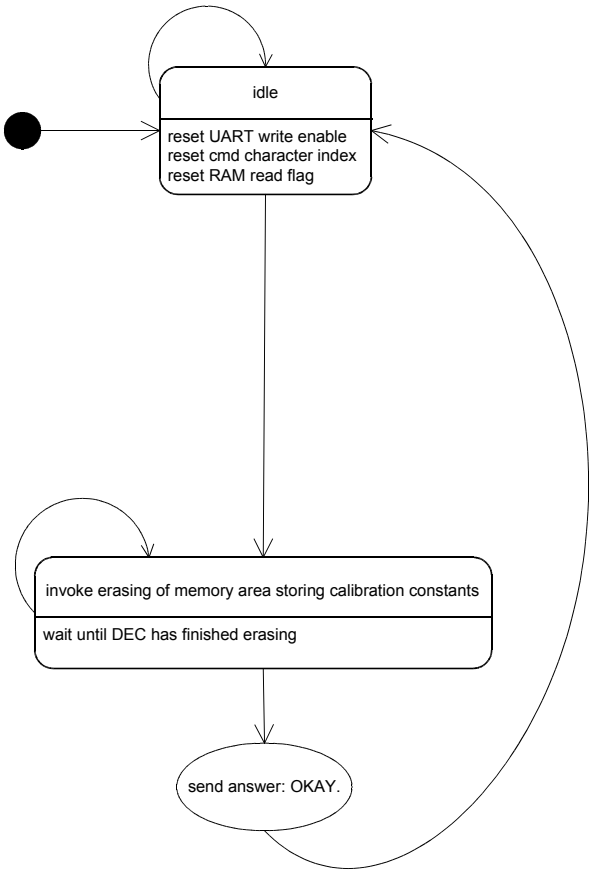


Figure 43: State chart SST - Switch off calibration

Subroutine: Switch to raw data output mode (RAW)

When data mode is switched to raw (*figure 44*), the behavior of the output process changes completely. Raw data means uncalibrated and uncorrected output. As illustrated in *chapter 5.1*, the amount of data per period doubles (26bit instead of 13bit) and accordingly data needs more cycles to be sent. So basically the ADC readout is just deserialized and passed through the FPGA.

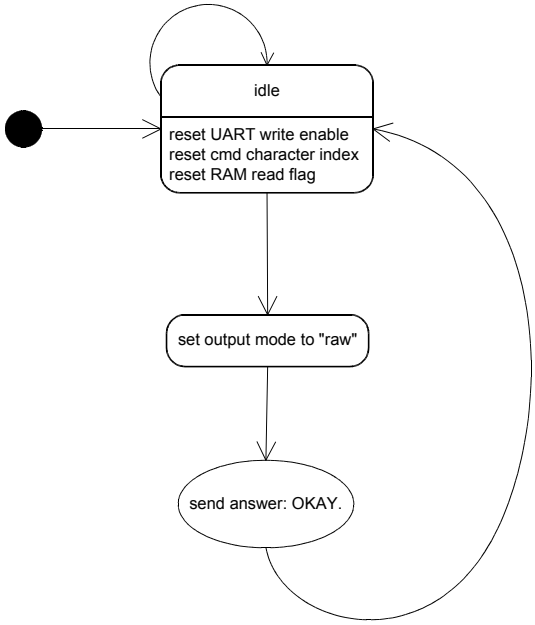


Figure 44: State chart SST - Switch to raw data mode

Subroutine: Send UART strobe signal / acknowledge

These two small subroutines (*figure 45*) are invoked every time a UART byte is received or ready to be transmitted, which happens quite frequently. The strobe and acknowledge signals are routed to the MiniUART subcomponent and are part of the Wishbone signal standard. They indicate that either received data is accepted or transmit data is ready to be sent.

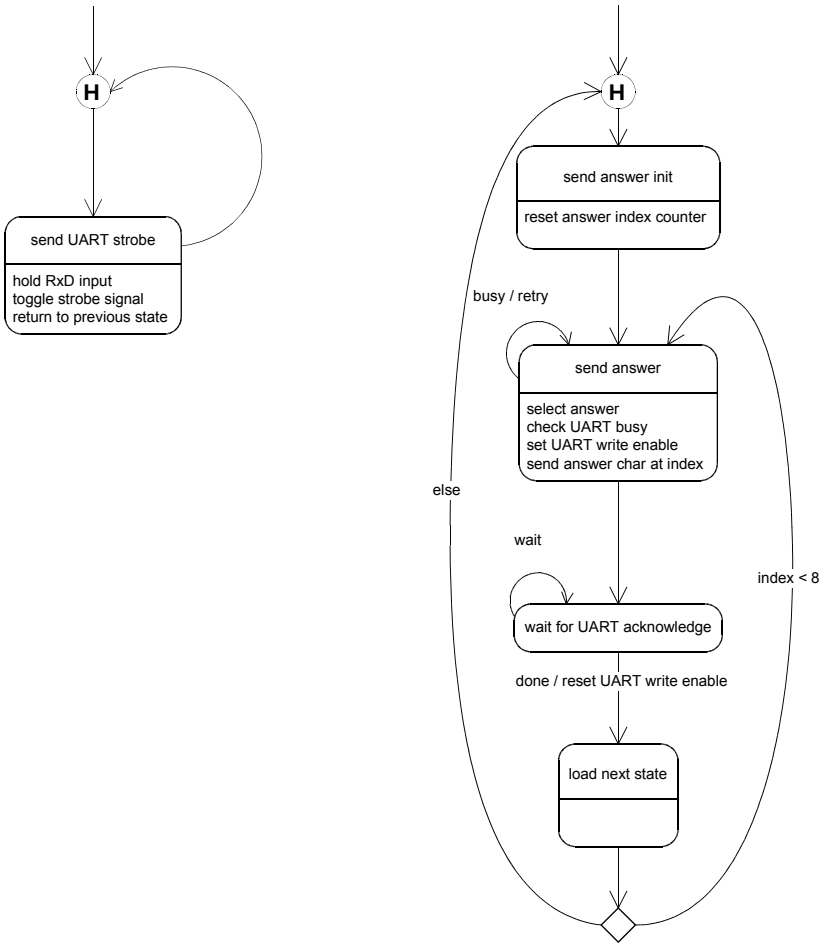


Figure 45: State chart SST - Send UART strobe / acknowledge

7.6.3 Pattern state machine (PST)

The pattern FSM operates the output of the CRIC-II control pattern (*figure 46*). Depending on the setting made by SST it starts / stops continuous or single generation of signal patterns previously loaded (command LDP) into the pattern RAM.

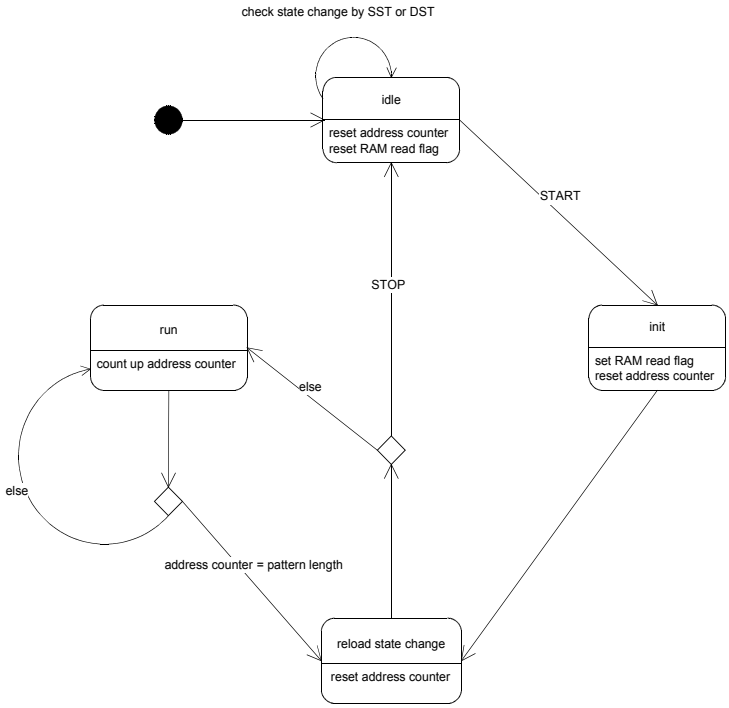


Figure 46: State chart PST - Generate control pattern for CRIC-II

7.6.4 Configuration state machine (CST)

CST or configuration state machine sends the configuration data delivered by SST to CRIC. As shown in *figure 47* it furthermore outputs the data and toggles the config clock signal every cycle until the counter reaches 58bit. So every rising edge one bit is accepted and after 2cycles×58bit=116 cycles CST returns to idle state.

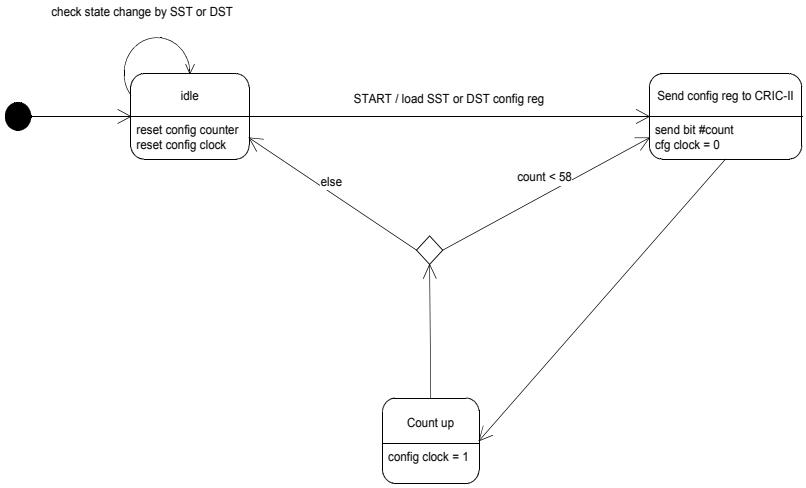


Figure 47: State chart CST - Send configuration data to CRIC-II

7.6.5 Calculation of calibration constants state machine (DST)

To calculate a new set of calibration constants DST has to be invoked by SST. *Figure 48* shows the data flow from top to bottom:

- Load configuration register by triggering CST to adjust the ADC to measure points in transfer function of cell *n*=6 to 13 (*see chapter 5.2.2*).
- Toss first measurements until ADC is stable (~1000).
- Take 1024 measurements and add them up.
- Calculate average.
- Branch to either calculate V_1-V_2 or V_3-V_4 or take next measurement and store present measurement temporarily.
- Store calculated calibration constant.
- Increment configuration counter:
  8 cells to calibrate × 4points per stage = 32 configuration sets.
- Repeat until all constants are determined

If output mode is switched to uncalibrated (by command NCL) DST is invoked with flag Clear RAM set active and all constants are deleted. This way no calculations are done at all and DST returns directly to idle state after erasing RAM.

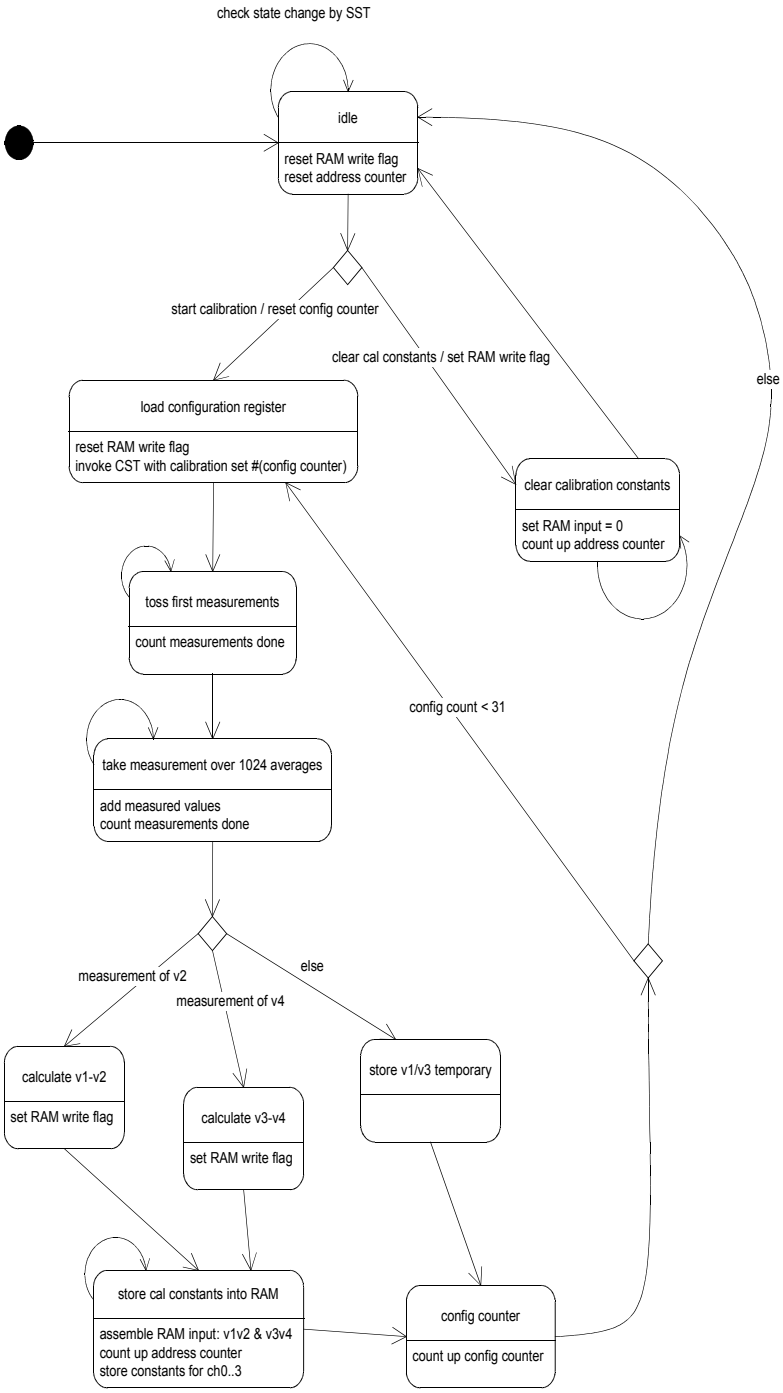


Figure 48: State chart DST - Calculation of calibration constants

7.6.6 FST: Final value state machine:

The calculation of the final ADC values requires one of the most complex FSMs of the design because it applies calibration on all acquired ADC measurements and uses both uncorrected and corrected readout data. As described in *chapter 5.2* FST reads the stored calibration constants and applies them according to the current cell's content. *Figure 49* shows that beginning with the LSB cell of all four channels ADC data is calibrated subsequently up to the MSB. After processing all eight calibratable cells the latch Data Valid is set and FST returns to idle state.

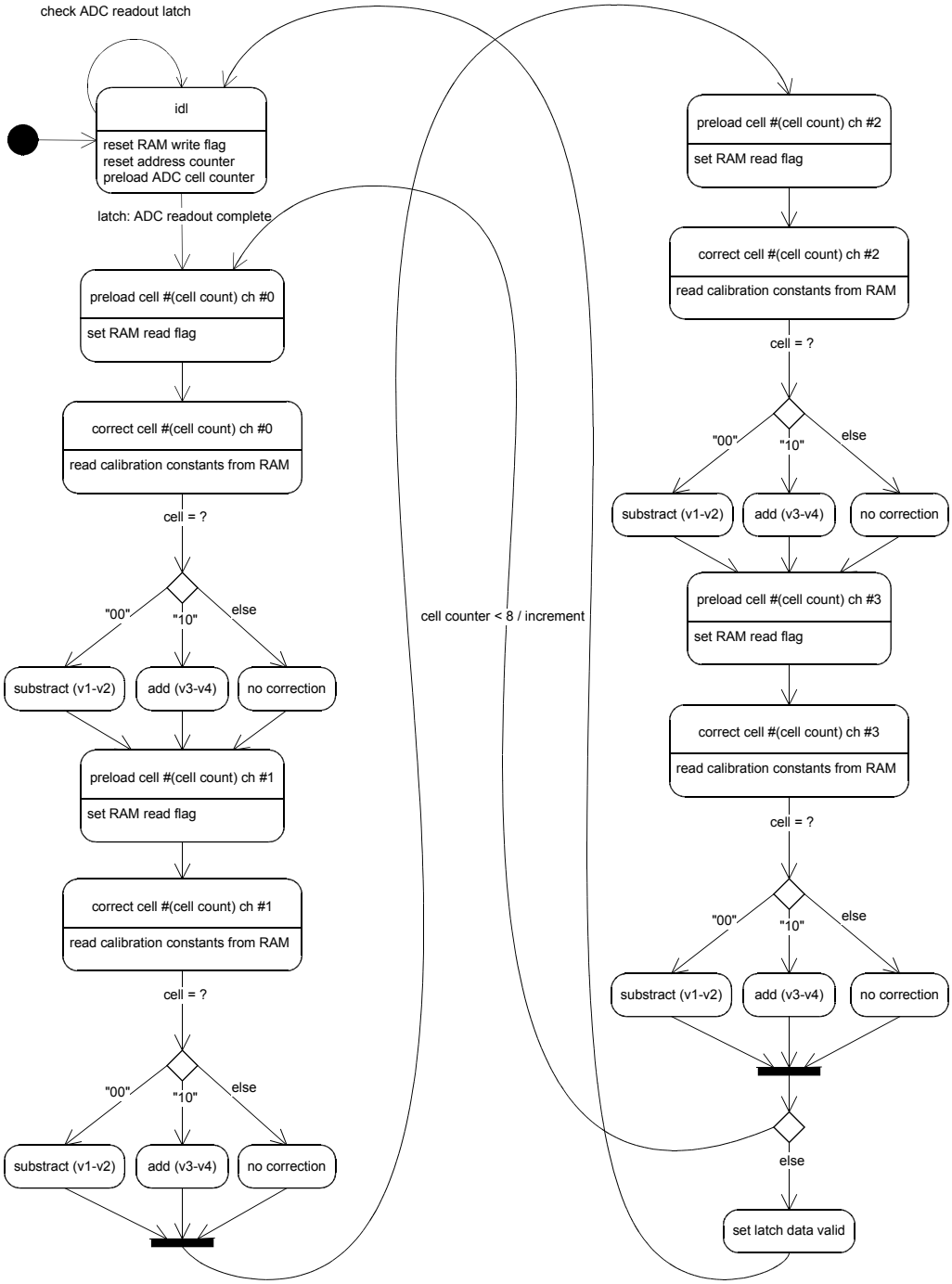


Figure 49: State chart FST - Applying the Digital Calibration

7.7 Design process and synthesis with Xilinx ISE

The entire design was compiled and synthesized with the free ISE WebPack synthesis software by Xilinx. It is restricted to CPLD and FPGAs with a maximum of 200kGates. For all larger FPGAs a commercial version has to be used that isn't free of charge. The ISE WebPack comprises the following tools:

- HDL & Abel editor (not used because not very good)
- Constraint editor PACE to assign resources like pins and hardware properties and to define area constraints
- Schematic editor to edit the compiled RTL logic
- Timing driven place and route synthesizer
- Floorplanner to check and modify resource assignments
- Simulator (*very* light version, only usable for small designs)
- IMPACT to access JTAG chain and to program the FPGA and the associated Flash PROM

8 Measurements

8.1 ADC specifications

Basically when looking at ADCs two characterizing parameter sets have to be discussed: The first are the AC domain specifications like SNR, SINAD, ENOB, etc., which aren't of interest in this context because they look at the ADC in terms of repeatability and not accuracy. The described methods affect accuracy , so the determination of DC domain specifications like differential and integral linearity are the most interesting parameters that have been measured extensively.

8.1.1 INL: Integral Non-linearity

The ADC's INL error is described as the deviation of an actual transfer function from a straight line (best fit line). Thus, the INL error magnitude directly depends on the position chosen for this straight line.

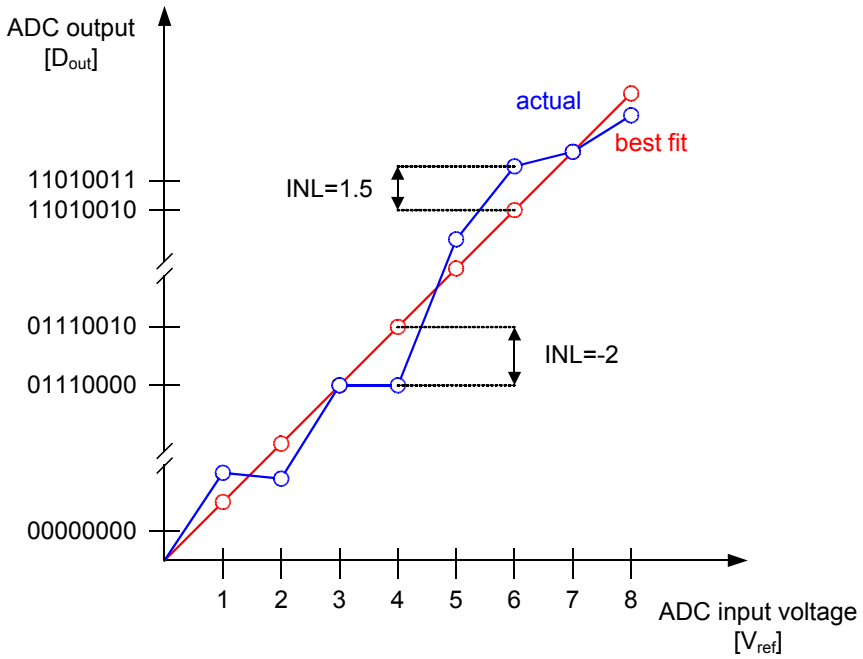


Figure 50: Integral Non-Linearity

The INL graph for an ADC displays the analog input (or input DAC value) on the x-axis and the digital output on the y-axis and is measured in LSB or percentage of full scale.

8.1.2 DNL: Differential Non-linearity

DNL error is defined as the difference between an actual step width and the ideal value of one LSB. For an ideal data converter, in which the differential non-linearity coincides with 0 LSB, each analog step equals 1 LSB.

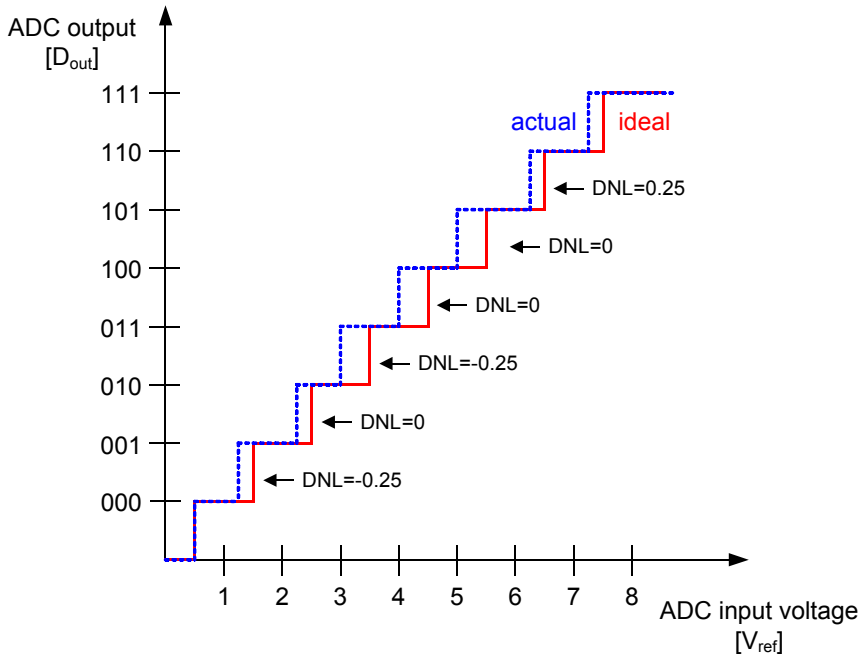


Figure 51:Differential Non-Linearity

$$1\,LSB = \frac{V_{FS}}{2^n}$$ with V_{FS} as the full-scale range and n as the ADC resolution

As shown in *figure 50* the DNL graph for an ADC displays the analog input on the x-axis and the digital output on the y-axis and is measured in LSB. To guarantee no missing codes and a monotonic transfer function an ADC's DNL must be greater than -1LSB.

8.2 INL measurements

To finally proof the effect of the implemented Digital Correction and particularly the Digital Calibration numerous measurements were taken. *Figure 53* shows an INL measurement without applying calibration, whereas *figure54* shows a measurement with calibration applied. The DAC values in both measurements refer to the onboard test DACs that are set to a range around half full-scale (*see chapter 6.2*).

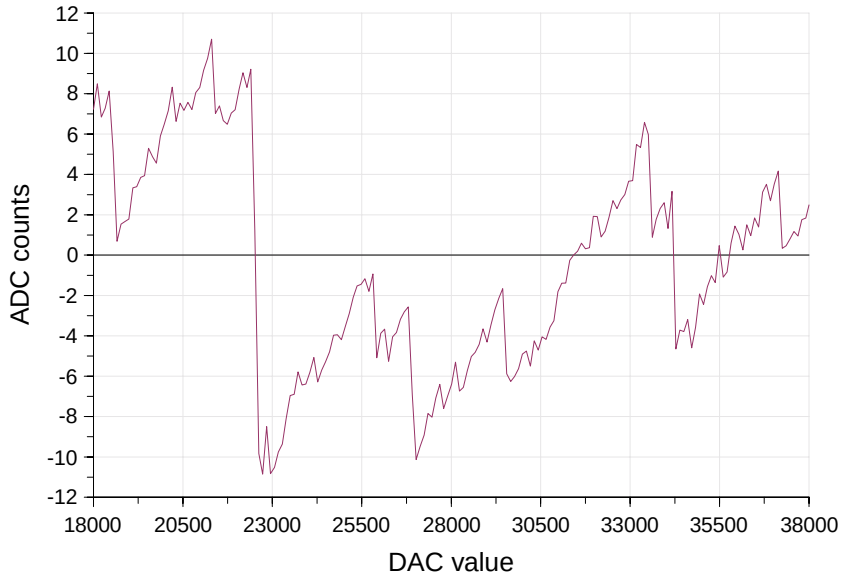


Figure 52: INL measurement uncalibrated

Now with calibration applied, full scale = 13bit = 8192.

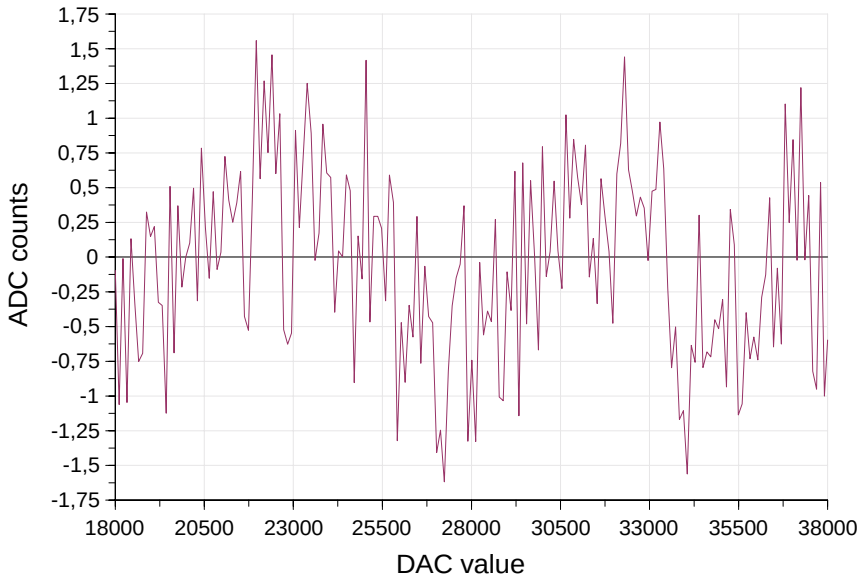


Figure 53: INL measurement calibrated

Conclusion: Improvement of about eight LSB!

8.3 DNL measurement

The following diagram shows the averaged DNL over the entire ADC input range:

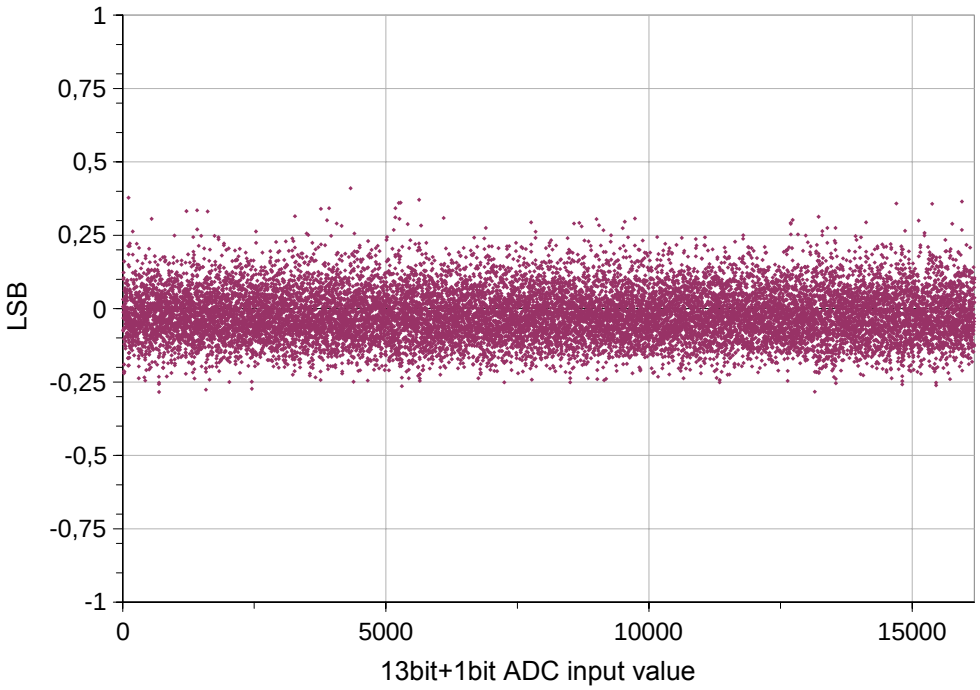


Figure 54: DNL measurement

Figure 54 is generated by averaging over 10000 measurements per point (→ADC input value). The input voltage consists of the DAC voltage (generated by onboard DAC) overlaid by a sawtooth waveform with $f_{saw} >> f_s$ (sampling frequency) and an amplitude of 1 LSB:

$$V_{in} = V_{DAC} + v_{saw}$$ (generated by external frequency synthesizer).

So V_{DAC} is incremented linearly while v_{saw} is kept at a constant frequency and amplitude. The frequency f_{saw} must not be a multiple of the sampling frequency, otherwise there wouldn't be a uniform distribution of all input values in the range of the current DAC setting plus 1 LSB.

Conclusion

The maximum DNL is about 0.4 LSB with an average deviation of about 0.1 to 0.2 LSB, which are quite typical magnitudes for pipeline ADCs.

8.4 Noise measurement

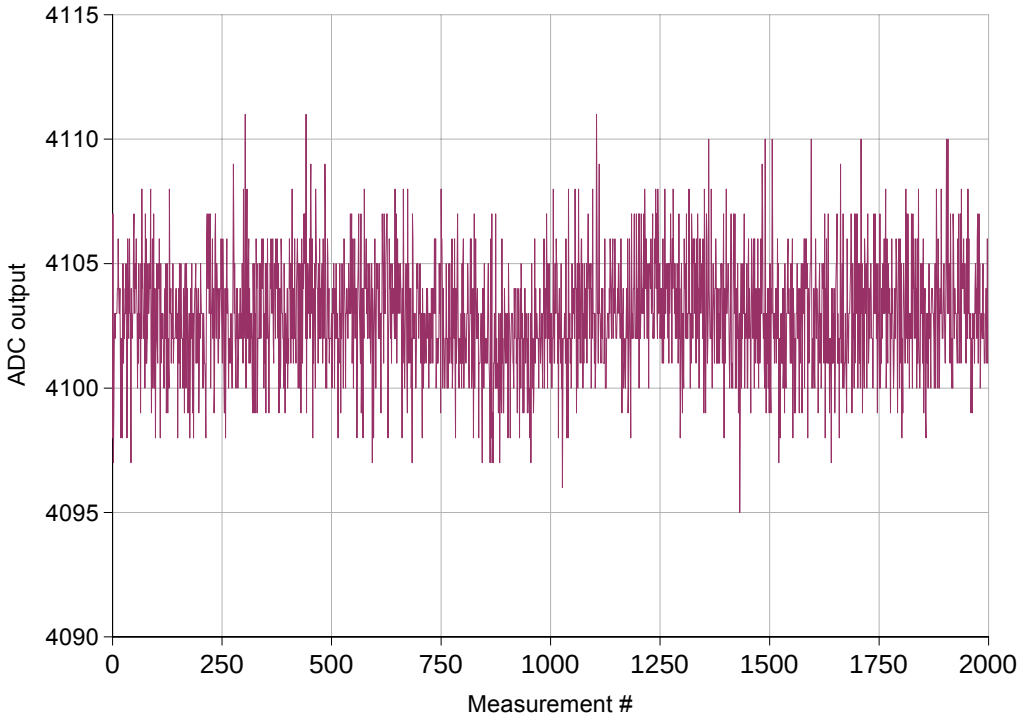


Figure 55: Noise measurement

Figure 52 shows a measurement of 2000 digitally corrected and calibrated ADC output values of channel 2. Input voltage is DC and around mid-range (~4096= full-scale/2). Since this measurement aims to find out about the ADC output dispersion, the exact input voltage isn't of central interest. Using the ADC output values it's possible to calculate the associated statistical parameters:

Average D_{out}:	μ_0 = 4102.92 LSB
Standard deviation:	σ = 2.28 LSB

Conclusion

Even at room temperature (25°C) the noise level is comparably low.

8.5 Statistical evaluation of the calibration constants

Since the calculated calibration constants should only be influenced by environmental factors like temperature and noise in general, a statistical evaluation of many (or at least some) calculation runs could be interesting. The following plot displays the accumulated standard deviation over ten calibration runs. It is calculated by specifying the distance of two adjacent standard deviations of each stage for all cells and channels (cell1=MSB):

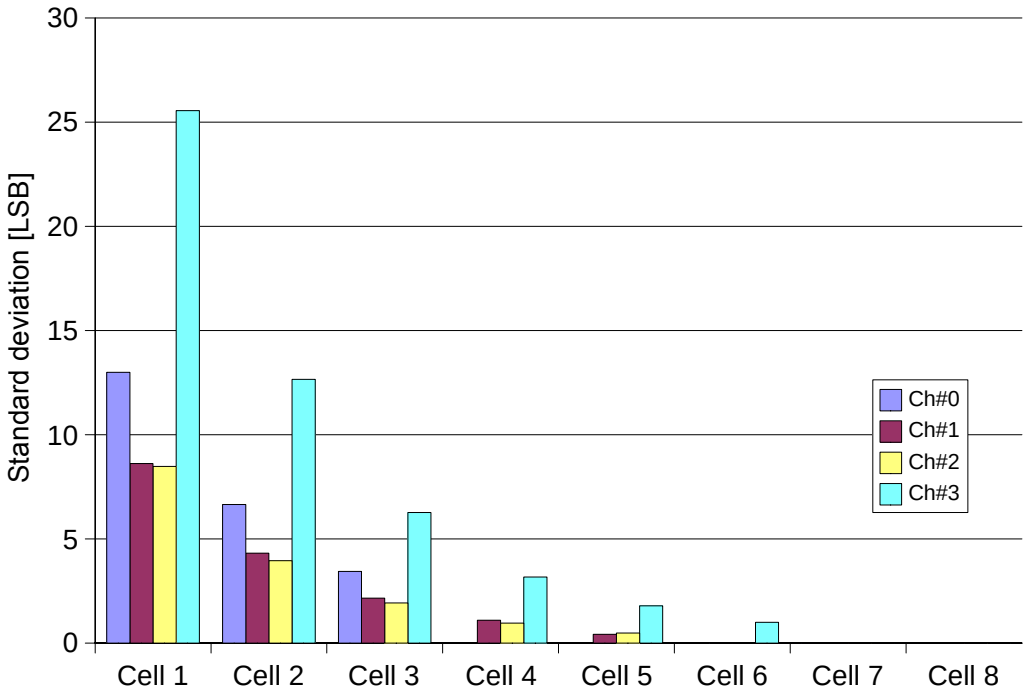


Figure 56: Standard deviation of calibration constants

Figure56 illustrates that a small error from one of the first LSB cells just propagates up to the MSB cell and hence doubles with every stage.

Conclusion

Apart from channel 3 the statistic is completely inconspicuous, assuming that V_1-V_2 and V_3-V_4 have about the same magnitude, what they actually do. Even for the higher deviation of channel 3 there's a reasonable explanation: Since additional test equipment was connected to this particular input at the time when the measurement was taken, additional noise was introduced into the system. Thus, the uncertainty of calculating stable constants for channel 3 increased.

9 Conclusion and future upgrades

9.1 Meeting the specs

After about six month of work on CRIC-II the main goal to realize an INL of ± 1 LSB at room temperature was reached . The synthesized VHDL code for driving and controlling the CRIC-II signals plus Digital Correction and Calibration was programmed into a Flash PROM and works now independently of the development system. Since CRIC-II is supposed to operate in space, one of the next steps will be to test the design for radiation hardness and to characterize the system in a cold environment by cooling it off with nitrogen.

9.2 The way to CRIC-III

Further development will be done to improve ADC accuracy and to reduce overall system noise. The proposed VHDL design is supposed to be implemented hard-wired on-chip as a part of CRIC-III and thus, will hopefully process the cosmological data of SNAP one day.

10 Appendix

10.1 VHDL code documentation

10.1.1 Pattern generation – Calibration – Main Control

TABLE OF CONTENTS

- /PatGenCalCtrl
- /PatGenCalCtrl/Architecture
- /PatGenCalCtrl/Architecture/Components
- /PatGenCalCtrl/Architecture/Concurrent_Statements
- /PatGenCalCtrl/Architecture/Process_SST
- /PatGenCalCtrl/Architecture/Process_PST
- /PatGenCalCtrl/Architecture/Process_ADCReadout
- /PatGenCalCtrl/Architecture/Process_CST
- /PatGenCalCtrl/Architecture/Process_AcqCntSum
- /PatGenCalCtrl/Architecture/Process_DST
- /PatGenCalCtrl/Architecture/Process_FST
- /PatGenCalCtrl/Architecture/Process_SignalOut

/PatGenCalCtrl

[top]

ABSTRACT

```
This VHDL code was designed to readout and test the CRIC-II ADC ASIC using a XILIN
Sprtan-IIE FPGA with 200kGates on an Avnet Spartan-IIE evaluation board.
The design performs the following tasks:


   - pattern generation to provide the essential readout control signals to
     CRIC-II and to the DAC switches on the test board
   - receive commands via UART interface from a serial terminal,
     send answers and data
   - control of CRIC-II configuration register
   - readout of CRIC-II output signals and output on a 32bit bus
   - digital error correction and calibration
   - switching of data modes to get raw or calibrated data and to provide other
     informations related to the measurement and the applied calibration
```

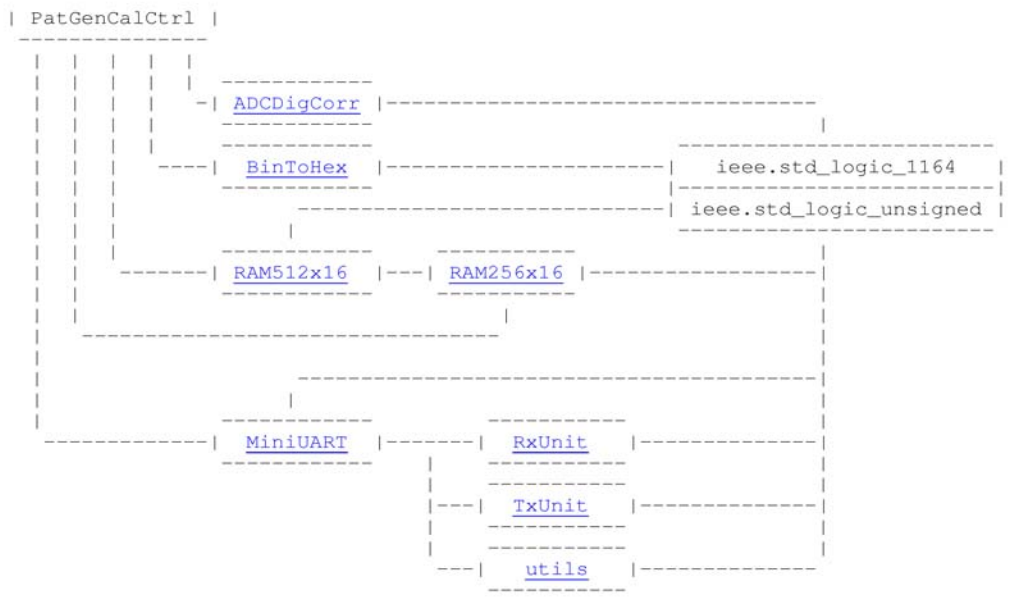
INPUT SIGNALS

```
signal     :[range] : description
-------------------------------------------------------------------------------
reset      :        : global reset, connected to DIP switch #1
clk        :        : global clock input, fxtal= 40MHz
ser_in     :        : serial UART input signal
adc_in     :[ 3..0] : input lines from CRIC-II
```

OUTPUT SIGNALS

```
signal     :[range] : description
-------------------------------------------------------------------------------
ser_out    :        : serial UART output signal
pat_out    :[31..0] : pattern, MSBs are to control switches, LSBs are CRIC-II sign
pat_out_n  :[31..0] : negative LVDS part of signal above
sig_out    :[31..0] : processed ADC signal to send to CRIC-II test software
ro_clk     :        : gated clock for readout of CRIC-II, derived from global cloc
ro_clk_n   :        : negative LVDS part of signal above
sig_latch  :        : indicates sig_out is valid
sig_latch_n:        : negative LVDS part of signal above
cfg_out    :        : signal connected to CRIC-II configuration register
cfg_clk    :        : clock for configuration register (shifting)
cfg_rst    :        : reset for configuration register
```

DEPENDENCIES



LEGEND

In text sections signal and variable names are mentioned using curly brackets, like {signalXY}.

TARGET

XILINX Spartan

SEE ALSO

XILINX application notes, XILINX Spartan docs & lib guide, CRIC-II test software d synthesis report file (pgcc.syr) for device utilization

AUTHOR

Markus Redelstab (redelstab@nanutech.com), Lawrence Berkeley Lab 2004

/PatGenCalCtrl/Architecture

[top]

ABSTRACT

This architecture contains nine processes. Five of them are explicit finite state machines (FSM). In particular these FSMs are:

- *SST: main state machine (serial UART state machine)*
  Controls the UART communication, i.e. receives commands from serial terminal (9600,8N1), stores the received pattern into the FPGA RAM, sends data and starts/stops other processes and state machines.
  command syntax: '#[magic word][COMMAND][cr]'
  All commands have to be sent in uppercase letters. If the FPGA UART interface expects further input, it gives you the answer 'STX...' (start transmission). The now following input has to be succeed by a checksum, which is the mod-256 sum of all bytes sent. The answer is either 'OKAY.' or 'ERR04' (checksum error). If a command isn't recogn at all, you get 'ERR01' (syntax error).
- *PST: pattern generation state machine*
  Runs the pattern received and stored by SST.
  This finite state machine (FSM) is controlled by SST and DST.

```
- CST: configuration state machine
       Sends 58bit of configuration data received by SST to CRIC-II.
       This FSM is controlled by SST and DST.
- DST: digital error correction state machine
       Calculates the calibration constants and stores them into the FPGA RAM.
       This FSM is controlled by SST.
- FST: final value state machine
       If calibration is switched on, it calculated the calibrated ADC values
       based on the calibration constants.
       This FSM isn't controlled by any other state machine it only triggers on
       new readout data.

In the following description of some state machines only main states are explained
If there are sub states, they usually fulfill a part of the task already described
the main state and have the same name followed by a number from 1 to n.
```

TYPE DEFINITIONS

```
st_type              : enumeration type for use with state variables
char                 : subtype representing an 8bit ASCII character
cmd_type             : array of char(0 to 3) is a four letter command
ans_type             : array of char(0 to 6) is a seven letter serial UART an
cfg_word             : subtype representing a std_logic_vector (57 DOWNTO  0)
                       {cfg_word} contains a complete set of CRIC-II config b
cfg_set              : array of cfg_word(0 to 31) infers a ROM inside the FPG
                       represents 32 sets of ADC adjustments,
                       32 config words = 8 cells to calibrate * value(v1..v4)
```

CONSTANTS

```
a_okay               : answer selector 'OKAY.'
a_stx                : answer selector 'STX...'
a_echk               : answer selector 'ERR04'
a_esyn               : answer selector 'ERR01'
a_ver                : answer selector 'VER2.1'
a_wait               : answer selector 'WAIT..'

cmd_mw               : command (cmd_type) containing magic word
cmd_rnc              : command (cmd_type) 'run continous'
cmd_rno              : command (cmd_type) 'run once'
cmd_stp              : command (cmd_type) 'stop pattern'
cmd_ldp              : command (cmd_type) 'load pattern'
cmd_cfg              : command (cmd_type) 'configuration'
cmd_ver              : command (cmd_type) 'version'
cmd_dec              : command (cmd_type) 'digital error correction'
cmd_ncl              : command (cmd_type) 'no calibration'
cmd_raw              : command (cmd_type) 'raw mode'

ans_ok               : answer (ans_type) '[cr]OKAY.[cr]'
ans_stx              : answer (ans_type) 'STX...[cr]'
ans_errc             : answer (ans_type) '[cr]ERR04[cr]'
ans_errs             : answer (ans_type) '[cr]ERR01[cr]'
ans_ver              : answer (ans_type) 'VER2.1[cr]'
ans_wait             : answer (ans_type) 'WAIT..[cr]'

ch_rrc               : type char '#'
ch_comma             : type char ','

ro_patpos            : position in the FPGA pattern RAM matrix, that holds
                       the signal to start the ADC readout process
pat_len              : length of pattern (400 timeslots)
                       400*(1/fxtal) = 400*25ns = 10us for one cycle
                       => fsmax= 100 kHz

cal_cfg              : Holds all 32 ADC adjustments for calibration and
                       infers a 32*64bit ROM (or read-only RAM).
```

SIGNALS

```
signal       :[range] : description
--------------------------------------------------------------------------------
```

```
state machines:

sst           :st_type : state register for main state machine (SST)
pst           :st_type : state register for pattern state machine (PST)
cst           :st_type : state register for configuration state machine (CST)
dst           :st_type : state register for DEC state machine (DST)
fst           :st_type : state register for final value state machine (FST)
[x]_nextby[y]:st_type : The state register of [x] reloads the value of this regist
                        (i.e. the next state) usually when returning to idle state
                        It is controlled by [y].
UART:

uart_sel      :[ 1..0] : mode selection, fixed to data mode "00"
uart_txd      :[ 7..0] : byte to transmit
uart_rxd      :[ 7..0] : byte to receive
uart_we       :        : write enable according to Wishbone IP core standard
uart_st       :        : strobe (Wishbone)
uart_ack      :        : acknowledge (Wishbone)
uart_ntxbsy   :        : UART not busy (is not used, inverted to signal below)
uart_txbusy   :        : TxD part of UART is busy (more convenient than above)
uart_rxnew    :        : Indicates new byte received.
checksum      :[ 7..0] : mod-256 sum of all bytes received/transmitted
ans_nr        :[ 3..0] : number of answer to be send (see constants)
idx           :[ 2..0] : index used to access command and answer arrays

pattern generator:

Since the pattern RAM is accessed by more than only one process, the usage of
seperate registers for read and write mode is mandatory (no multisource!)

rpat1_di      :[15..0] : data in pattern RAM #1
rpat1_do      :[15..0] : data out pattern RAM #1
rpat2_di      :[15..0] : data in pattern RAM #2
rpat2_do      :[15..0] : data out pattern RAM #2
rpat_write    :        : flag, sets RAM to write mode
rpat_read     :        : flag, sets RAM to read mode
rpat_mode     :        : holding the result from combining the two signals above (a
rpat_addr_r   :[ 8..0] : address during read mode
rpat_addr_w   :[ 8..0] : address during write mode
rpat_addr     :[ 8..0] : address during read/write multiplexed according to rpat_mo

ADC readout:

out_mode      :        : Switches signal output of FPGA between calibrated and raw
                         mode. Raw mode (1, default) outputs only deserialized ADC
                         data, whereas cal mode (0) provides entirely calibrated da
sig_int0..3   :[ 6..0] : shift register used for deserializing CRIC-II data ch#0..3
ro_cnt        :[ 2..0] : Counts received ADC bits up to the 7th and triggers latch_
ro_start      :        : triggers new readout cycle (part of pattern)
ro_reg        :        : signal above delayed for one clock
latch_raw     :[ 6..0] : latch used to signal completed 7bit readout cycle (see abo
                         Shifts "11" to the left => 2 clks signal width / 5 clks de
ro_comp       :[ 1..0] : Counts completed sets of 7bit ADC data => 7bit*4cycles= 28
adcro_ch0..3  :[27..0] : assembled complete set of 28bit for ch#0..3
adc_corr0..3  :[12..0] : data, digitally corrected by component ADCDigCorr
ro_clk_r      :        : gated global clock out, active during ro_start = high
ro_clk_r_n    :        : neg. LVDS part of signal above

configuration register:

cfg_regbys    :[57..0] : Contains configuration to be send, set by SST-FSM
cfg_regbyd    :[57..0] : Contains configuration to be send, set by DST-FSM
cfg_reg       :[57..0] : configuration that's sent after invoking CST-FSM
                         multiplex of signals mentioned above, controlled by async
                         triggering SST/DST states (see concurrent statements)
cfg_cnt       :[ 5..0] : Counts down sent configuration bits in CST-FSM
cfg_idx       :[ 5..0] : Indicates index position in {cfg_regbys} while receiving
                         configuration bytes via UART in SST-FSM

Summing & Averaging of acquired and calibrated ADC data:
```

```
acqcnt_en     :        : enable for this process
acq_cnt       :[10..0] : number of ADC values to be acquired
sum_ch0..3    :[23..0] : sum of ADC values ch#0..3

Calculation of calibration constants:
                                                          1 3 /
To the right you can see the assignment of signal names  /|/|/
according to the points in ADC transfer function (v1..v4):  / 2 4

tmp_ch0..3    :[23..0] : temporary register for substracting sums (1024x) of (v1-v2
                         (v3-v4) for ch#0..3
v1v2_ch0..3   :[23..0] : used to calculate final values of (v1-v2) after summing (1
                         substracting (v1-v2), rounding and averaging (ch#0..3)
                         Due to rounding {v1v2_ch0..3(9 downto 0)} is tossed.
v3v4_ch0..3   :[23..0] : same as above but for (v3-v4)
dec_cfgcnt    :[ 4..0] : counter for loading 32 ADC configuration for 8 cells to be
                         calibrated. 32 = 8cells * 4TFP (transfer function points)
dec_pos       :[ 1..0] : input signal to BinToHex, points to the digit that is to b
                         transcoded to hex ASCII (see concurrent statements).
dec_hex0..1   :[ 7..0] : hex ASCII representations calculated by BinToHex to send t
                         serial terminal (usually CRIC-II test software)
mask_off      :        : Switches off mask that's used to get rid of upper cells
                         which are currently not under calibration. This flag is
                         set asynchronously and passed to ADCDigCorr.

Calculation of final ADC values:

f_cellcnt     :[ 4..0] : Counts index of {adcro_ch0..3} from 11 to 25 and provides
                         pointer on the cell (2bit) that is about to be corrected w
                         the appropriate calibration constant.
adc_final0..3:[12..0] : That's the final result after adding all calibration const
                         for each of the eight cells.
cell          :[ 1..0] : cell that is currently processed (preloaded for next state
latch_cal     :[23..0] : This latch is generated after all calculations are done an
                         data is ready to be sent out. It is loaded with a bit sche
                         containing two seperate latch peaks and some time between.
```

/PatGenCalCtrl/Architecture/Components

[top]

INSTANCES

```
serial interface [ser_interface]:

This instance of the MiniUART IP core provides all necessary functions to implemen
a full duplex serial interace without hardware handshake. It complies with the
Wishbone IP core standard. The generic {brdivisor} is used to adjust the baudrate.

pattern RAM 1/2 [patram1..2]:

Two instances of RAM512x16 are used to have parallel access to 2*16 = 32bit of pat
data. The RAM is written by SST and read by PST. The pattern uses 400*32bit = 1600

calibration RAM 1/2 [calram1..2]:

Two instances of RAM256x16 are used to have parallel access to 2*16 bit calibratio
constants. Therefore you can read both calibration constants for a particular cell
once, which reduces the time needed to complete the calculation of the final value

ADC digital correction [digcorr0..3]:

The component ADCDigCorr is instantiated four times to have corrected ADC data ava
at one time for all four channels. The MSB of input vector {cell} is routed to
{mask_off} to switch off masking completely, whereas the remaining part is connect
to {dec_cfgcnt(4 downto 2)} to turn off the upper cells during calculation of the
calibration constants of the lower cells. Because the signal {dec_cfgcnt} ranges
from 0 to 31 and we toss bit0/1, it passes a value devided by four, i.e. 0..7, whi
the number of the cell under calibration. The two LSBs are used to point to the
```

part of {adc_corr} that is to be transcoded by BinToHex for later transmission to terminal via UART (see below). The vector {adc_in} is routed to {adcro_ch0..3(25 d which means that the LSB of the LSB cell is tossed. The two slope bits 27/26 aren' needed during digital correction and also tossed.

binary to hex transcoder [bintohex1..2]:

As mentioned above this component (instantiated two times) provides the calibratio constants in their hexadecimal ASCII representation to pass them to MiniUART.

buffers:

Due to the high transmission speed on some lines and to avoid cable length issues, some signals are configured as LVDS or at least as LVTTL with a pos/neg pair. To u these features of the Spartan-2e you have to instantiate particular intrinsic comp that are recognized by the synthesizer as a black box macro.

SEE ALSO

XLINX library guide, MiniUART docs, RAM512x16, RAM256x16, ADCDigCorr, BinToHex

---

/PatGenCalCtrl/Architecture/Concurrent_Statements

[top]

ABSTRACT

The following asynchronous assignments have been made and are listed as follows.

SOURCE

```
   uart_sel <= "00";                              -- UART mode: only receive/t
   uart_txbusy <= NOT uart_ntxbsy;                -- inverting for convenience
-- ---------------------------------------------------------------------------
   rpat_mode <=                                   -- pattern RAM mode select a
      '0' WHEN rpat_read  = '1' ELSE              -- to read/write flags
      '1' WHEN rpat_write = '1'                   -- default: read (0)
      ELSE '0';
   rpat_addr <=                                   -- RTFP address select accor
      rpat_addr_w WHEN rpat_write = '1' ELSE      -- to read/write address fla
      rpat_addr_r WHEN rpat_read  = '1'           -- default: 0x00
      ELSE X"00"&'0';
-- ---------------------------------------------------------------------------
   ro_start <=                                    -- trigger ADC readout by
      patout_buf(conv_integer(ro_patpos));        -- specific pattern signal
   ro_clk_r <= clk
      WHEN (ro_start = '1') ELSE '0';             -- switch gated readout cloc
-- ---------------------------------------------------------------------------
   sig_latch <=                                   -- LVDS routing of delayed l
      latch_raw(6) WHEN out_mode = '1'            -- & switching of raw/cal ou
      ELSE latch_cal(23);
   sig_latch_n <=                                 -- same as above
      NOT latch_raw(6) WHEN out_mode = '1'        -- but neg LVDS part
      ELSE NOT latch_cal(23);
-- ---------------------------------------------------------------------------
   rtfp_mode <=                                   -- cal const RAM mode select
      '0' WHEN rtfp_read_f  = '1' ELSE            -- according to read/write f
      '0' WHEN rtfp_read_s  = '1' ELSE            -- default: read (0)
      '1' WHEN rtfp_write_d = '1'
      ELSE '0';                                   -- RTFP address range: 0..31

   rtfp_addr <=                                   -- RTFP address select
      rtfp_addr_w   WHEN rtfp_write_d= '1' ELSE   -- default: 0x00
      rtfp_addr_r_f WHEN rtfp_read_f = '1' ELSE
      '0' & rtfp_addr_r_s(7 downto 1)             -- bit0 is used to mux
         WHEN rtfp_read_s  = '1'                  -- dec_hex1/2 to rtfp_out
      ELSE X"00";                                 -- during UART transmission
-- ---------------------------------------------------------------------------
   mask_off <= '1' WHEN dst = idle ELSE '0';      -- mask only during cal cons
-- ---------------------------------------------------------------------------
```

```
   cst_nextbys <= send_cfg_reg_1                -- starts transmission of co
       WHEN sst = rec_send_cfg_trans_trig       -- controlled by SST
      ELSE idle;
-- ---------------------------------------------------------------------------
   cst_nextbyd <= send_cfg_reg_1                -- starts transmission of co
        WHEN dst = deccalc_load_cfg_reg_trig    -- controlled by DST
      ELSE idle;
-- ---------------------------------------------------------------------------
   pst_nextbyd <=                               -- starts pattern generation
      patgen_init WHEN dst /= idle              -- controlled by DST
      ELSE idle;
-- ---------------------------------------------------------------------------
   led(0) <=                                    -- debugging info:
      '1' WHEN sst /= idle ELSE '0';            -- indicates which FSMs
   led(1) <=                                    -- are running = not idle
      '1' WHEN pst /= idle ELSE '0';
   led(2) <=
      '1' WHEN cst /= idle ELSE '0';
   led(3) <=
      '1' WHEN dst /= idle ELSE '0';
   led(4) <=
      '1' WHEN fst /= idle ELSE '0';
   led(6 downto 5) <= "00";                     -- unused
   led(7) <= out_mode;                          -- indicates raw/cal mode
```

/PatGenCalCtrl/Architecture/Process_SST

[top]

ABSTRACT

This process provides the main state machine that controls most of the other FSMs and processes. Basically it is capable of receiving commands and sending answers via serial UART. Therefore it's the origin for almost all actions being initiated.

STATES

```
det_cmd_magicword      : detection of the magic word at the beginning of transm
det_cmd_1st_char       : After recognition of the 1st char, branch to det_cmd_[
                         examine the remaining characters. All commands have to
                         different in the 1st character.
det_cmd_run_cont       : Examines all chars of command 'run continously'(MRN)
det_cmd_run_once       : Examines all chars of command 'run once'(RNO)
det_cmd_stop           : Examines all chars of command 'stop pattern'(STP)
det_cmd_load_pattern   : Examines all chars of command 'load pattern'(LDP)
det_cmd_config_register: Examines all chars of command 'configuration'(CFG)
det_cmd_version        : Examines all chars of command 'version'(VER)
det_cmd_dig_error_corr : Examines all chars of command 'digital error correctio
det_cmd_no_cal         : Examines all chars of command 'no calibration'(NCL)
det_cmd_raw_mode       : Examines all chars of command 'raw'(RAW)
-------------------------------------------------------------------------------
send_answer            : Sends answers to serial terminal according to {ans_nr}
-------------------------------------------------------------------------------
record pattern         : Stores pattern into FPGA RAM received from serial term
                         The pattern is secured by a checksum.
-------------------------------------------------------------------------------
rec_send_cfg_uart      : Receives new content (58bit) for the CRIC-II configura
                         register and triggers CST-FSM to send it to CRIC-II.
-------------------------------------------------------------------------------
dec_mem_erase          : Starts erasing of FPGA RAM that stores calibration con
dec_calc               : Starts calculation of calibration constants.
dec_read_wait          : Waits until RAM read cycle is finished and read values
                         transcoded to hexadecimal ASCII characters
dec_send_uart          : Sends hexadecimal ASCII calibration constants to seria
                         terminal, i.e. CRIC-II test software.
-------------------------------------------------------------------------------
send_uart_strobe       : Toggles strobe signal to the Mini-UART after receiving
                         a new character.
-------------------------------------------------------------------------------
```

```
    idle                    : Nothing to do, waiting to be awaked
```

SENSITIVITY LIST

```
clk, reset
```

/PatGenCalCtrl/Architecture/Process_PST

[top]

ABSTRACT

```
Initializes pattern output and controls address counter of pattern RAM.
PST is invoked by either SST or DST.
```

STATES

```
    patgen_init             : Initializes pattern generation coming from IDLE.
    patgen_reload           : Reloads the registers/counters after completing a cycl
    patgen_run              : Sets the pattern generation signal outputs after loadi
                              their values from the FPGA RAM.
    idle                    : Nothing to do, waiting to be awaked
```

SENSITIVITY LIST

```
clk, reset
```

/PatGenCalCtrl/Architecture/Process_ADCReadout

[top]

ABSTRACT

```
This process is triggered by {ro_start} and assembles 28bit ADC readout data.
Every {ro_start} cycle we get 4bit, i.e. 1bit each channel. So after seven cycles
we have 4*7bit in the shift registers {sig_int0..3}. Depending on the {outmode}
setting these 28bit are sent in raw format immediately or further assembled
to 4*28bit (later processed by FST-FSM). The finally assembled raw data is held by
{adcro_ch0..3} and signaled valid by {latch_raw} trigger.
```

SENSITIVITY LIST

```
clk, reset
```

/PatGenCalCtrl/Architecture/Process_CST

[top]

ABSTRACT

```
CST or Configuration STate machine is in charge of sending config data to CRIC-II.
This data is either received by SST via UART or read out from ROM during calculati
of calibration constants.
```

STATES

```
    send_cfg_reg            : Sends the received configuration register values to CR
    idle                    : Nothing to do, waiting to be awaked
```

SENSITIVITY LIST

```
clk, reset
```

/PatGenCalCtrl/Architecture/Process_AcqCntSum

[top]

ABSTRACT

While calculating the calibration constants for each ADC cell and measuring v1 to v4 of the ADC transfer function, there's a lot of averaging going on. That's what this process is about. Everytime when there's a new set of ADC values available (triggered by {lach_cal}) these values are added up and counted by {acq_cnt}. It can be reset and disabled by setting {acqcnt_en} to high.

STATES

send_cfg_reg	: Sends the received configuration register values to CR
idle	: Nothing to do, waiting to be awaked

SENSITIVITY LIST

clk, reset

/PatGenCalCtrl/Architecture/Process_DST

[top]

ABSTRACT

Everytime this process is invoked by SST it calculates a new set of calibration constants or at least erases the memory area storing them. But before you start a calibration run, please make sure you loaded the right pattern because the driving of some readout signals is different during calibration.
The complete DEC process works like this:
After erasing is finished, it loads one of 32 configuration sets and sends it to CRIC-II to switch on/off ADC cell 1..8 for v1..v4 (i.e. 8*4=32 configurations). Th now following measurement is done by first tossing some data to ensure that everything has stabilized and second by taking 1024 averages to reduce noise. At last the values v1-v2 and v3-v4 are substracted, rounded and finally stored into TFP RAM (TFP = Transfer Function Point).

STATES

deccalc_erase_mem	: Erases the FPGA RAM areas storing calibration constant
deccalc_load_cfg_reg	: Loads configuration register from read-only FPGA RAM t calibrate ADC cell 1..8 and triggers CST to send new content to CRIC-II.
deccalc_toss1st	: Tosses the first 2000 values to ensure that the DACs o the CRIC-II test board are stable.
deccalc_tfp	: Calculates the correction values by getting the value the transfer function in its four turning points.
deccalc_store	: Stores the calibration constants calculated above into the FPGA RAM.
idle	: Nothing to do, waiting to be awaked

SENSITIVITY LIST

clk, reset

/PatGenCalCtrl/Architecture/Process_FST

[top]

ABSTRACT

FST (Final value statemachine) acts as core calibration process. It examines ADC cell 1..8 to decide whether and which calibration constant has to be applied. That means, if cell=="00" => -(v1-v2), "10" => +(v3-v4), else => do nothing. In the end calibrated and digitally corrected ADC values are provided in {adc_final0..3} and signaled valid by {latch_cal} trigger.

STATES

final_init_ch0..3	: Sets adress select of FPGA RAM to get calibration cons for correcting ADC cell 1..8 of ADC channel #0..3.

```
                          Preloads cell under examination (1..8).
    final_corr_ch0..3     : Corrects ADC value according to preloaded cell by
                          substracting or adding appropriate calibration constan
    final_next            : Counts up cell number 1..8.
    idle                  : Nothing to do, waiting to be awaked
```

SENSITIVITY LIST

```
clk, reset
```

/PatGenCalCtrl/Architecture/Process_SignalOut

[top]

ABSTRACT

```
Depending on {out_mode} this process outputs either raw or calibrated ADC data. If
{out_mode} is set to calibrated, it triggers on {latch_cal} and switches between
ch#01/1 and ch#2/3 since we have a 32bit bus and the amount of data is
2*13bit data + 2*2bit slope + 2*1bit parity= 32bit.
So a complete set of data is transmitted after two cycles or after shifting
{latch_cal} 12 times. For the case of sending raw data it's much easier because
ADC data is sent out without being processed further, so a complete set is done
after four cycles of 28bit. Therefore each cycle contains 7bit data of all four
channels beginning with the LSB, the remaining 4*1bit on the top (MSB) are always
set to '0' to complete 32bit.
```

SENSITIVITY LIST

```
clk, reset
```

10.1.2 ADC Digital Correction

TABLE OF CONTENTS

- /ADCDigCorr
- /ADCDigCorr/Architecture
- /ADCDigCorr/Architecture/Components
- /ADCDigCorr/Architecture/Process_DigCorr

/ADCDigCorr

[top]

ABSTRACT

```
Applies digital correction and maskes out MSB bits during calculation
of calibration constants.
```

INPUT SIGNALS

```
signal  :[range] :  description
-------------------------------------------------------------------------------
reset   :        :  global reset
clk     :        :  global clock input
adc_in  :[24..0] :  ADC input vector without slope bits and LSB of LSB cell
cell    :[ 3..0] :  0..7  => selection of cell number during calibration
cell    :[ 3..0] :  8..15 => no mask used at all
```

OUTPUT SIGNALS

```
signal  :[range] :  description
-------------------------------------------------------------------------------
adc_corr:[12..0] :  corrected ADC value output vector
```

DEPENDENCIES

```
ieee.std_logic_1164.ALL, ieee.std_logic_unsigned.ALL
```

TARGET

```
XILINX Spartan
```

AUTHOR

```
Markus Redelstab (redelstab@nanutech.com), Lawrence Berkeley Lab 2004
```

/ADCDigCorr/Architecture

[top]

ABSTRACT

```
Applies digital correction and maskes out MSB cells during calculation
of calibration constants.
```

/ADCDigCorr/Architecture/Components

[top]

ABSTRACT

```
- none -
```

10.1.3 Binary to Hex Transcoder

TABLE OF CONTENTS

- /BinToHex
- /BinToHex/Architecture
- /BinToHex/Architecture/Components
- /BinToHex/Architecture/Process_DigCorr

/BinToHex

[top]

ABSTRACT

Transcodes 16bit binary input vector to 8bit hexadecimal ASCII representation according to position given by pos ranging from 3 down to 0 (0=LSB).

INPUT SIGNALS

signal	:[range] :	description
reset	:	: global reset
clk	:	: global clock input
bin	:[15..0]	: binary input value
pos	:[1..0]	: position in 4 digit hex representation (0=LSB)

OUTPUT SIGNALS

signal	:[range] :	description
hex	:[7..0]	: output vector containing ASCII encoded hex character

DEPENDENCIES

ieee.std_logic_1164.ALL, ieee.std_logic_unsigned.ALL

TARGET

XILINX Spartan

AUTHOR

Markus Redelstab (redelstab@nanutech.com), Lawrence Berkeley Lab 2004

/BinToHex/Architecture

[top]

ABSTRACT

Transcodes 16bit binary input vector to 8bit hexadecimal ASCII representation according to position given by {pos} ranging from 3 down to 0 (0=LSB).

SIGNALS

signal	:[range] :	description
val	:[3..0]	: value at given position in binary input vector

/BinToHex/Architecture/Components

[top]

10.1.4 RAM 256x16

TABLE OF CONTENTS

- /RAM256x16
- /RAM256x16/Architecture
- /RAM256x16/Architecture/Components
- /RAM256x16/Architecture/Process_RAM256x16

/RAM256x16

[top]

ABSTRACT

Synthesizes a 256x16 block RAM on XILINX Spartan.

INPUT SIGNALS

```
signal   :[range] :  description
-------------------------------------------------------------------------------
reset    :        :  global reset
clk      :        :  global clock input
addr     :[ 7..0] :  address
di       :[15..0] :  data in
en       :        :  "chip" enable
we       :        :  write enable
```

OUTPUT SIGNALS

```
signal   :[range] :  description
-------------------------------------------------------------------------------
do       :[15..0] :  data out
```

GENERICS

```
signal   :[range] :  description
-------------------------------------------------------------------------------
init_val:[string]:  Initialization value for first RAM allocation
```

DEPENDENCIES

ieee.std_logic_1164.ALL, ieee.std_logic_unsigned.ALL

TARGET

XILINX Spartan

SEE ALSO

XILINX documentation & application notes

AUTHOR

Markus Redelstab (redelstab@nanutech.com), Lawrence Berkeley Lab 2004

/RAM256x16/Architecture

[top]

ABSTRACT

Synthesizes a 256x16 block RAM on XILINX Spartan.

TYPE DEFINITIONS

10.1.5 RAM 512x16

TABLE OF CONTENTS

- /RAM512x16
- /RAM512x16/Architecture
- /RAM512x16/Architecture/Components
- /RAM512x16/Architecture/Process_RAM512x16

/RAM512x16

[top]

ABSTRACT

Synthesizes a 512x16 block RAM on XILINX Spartan.

INPUT SIGNALS

signal	:[range] :	description
reset	:	: global reset
clk	:	: global clock input
addr	:[8..0]	: address
di	:[15..0]	: data in
mode	:	: 0 => read, 1 => write

OUTPUT SIGNALS

signal	:[range] :	description
do	:[15..0]	: data out

DEPENDENCIES

ieee.std_logic_1164.ALL, ieee.std_logic_unsigned.ALL,
RAM256x16

TARGET

XILINX Spartan

SEE ALSO

XILINX documentation & application notes,
documentation about RAM256x16

AUTHOR

Markus Redelstab (redelstab@nanutech.com), Lawrence Berkeley Lab 2004

/RAM512x16/Architecture

[top]

ABSTRACT

Synthesizes a 512x16 block RAM on XILINX Spartan out of two 256x16 RAMs.

SIGNALS

signal	:[range] :	description
do1	:[15..0]	: data out RAM #1
do2	:[15..0]	: data out RAM #2
en1	:	: "chip" enable RAM #1

10.1.6 Mini UART

TABLE OF CONTENTS

- /MiniUART

/MiniUART

[top]

ABSTRACT

Uart (Universal Asynchronous Receiver Transmitter) for SoC.
Wishbone compatible.

DEPENDENCIES

ieee.std_logic_1164.ALL,
Rxunit.vhd, Txunit.vhd, utils.vhd

TARGET

XILINX Spartan

COPYRIGHT

This core adheres to the GNU public license

SEE ALSO

MiniUART documentation

AUTHOR

Philippe CARTON (philippe.carton2@libertysurf.fr)

10.2 Example: Synthesis of Digital Correction

To give an idea of the design process the following chapter describes the schematic of a synthesized RTL logic and the corresponding VHDL code (*see chapter 5.1*)

On top of *figure57* there's a multiplexer selecting the uncorrected ADC input according to the cell[2 downto 0] mask selection (*see chapter 10.1 for more details*). The binary combinations for the cell signal are ranging from [000] to [111]. Since cell[3] basically stands for an on/off switch for masking, it's compiled as control input of the subsequent MUX to either select unmasked or masked data. The output of this stage is connected to a simple full-adder to do the essential Digital Correction by inter-stage summing of the ADC stages. The last stage of any synchronous components is always an output buffer.

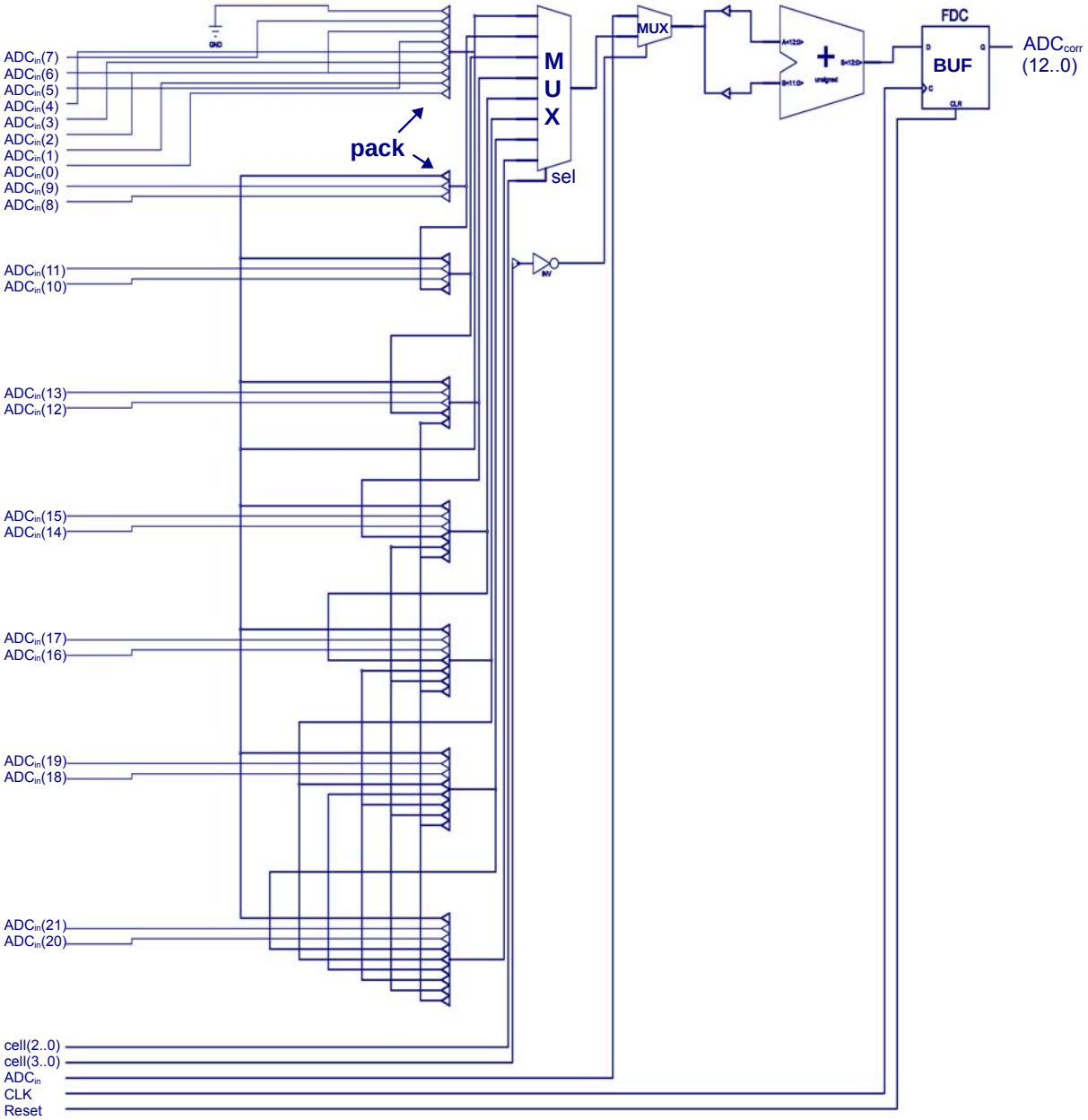


Figure 57: Schematic of Digital Correction

Following there's the corresponding VHDL implementation:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ADCDigCorr IS
  port (
    reset    : IN  std_logic;
    clk      : IN  std_logic;
    adc_in   : IN  std_logic_vector (24 DOWNTO 0);
    cell     : IN  std_logic_vector (3  DOWNTO 0);
    adc_corr : OUT std_logic_vector (12 DOWNTO 0)
  );
END ADCDigCorr;


ARCHITECTURE Arch_ADCDigCorr OF ADCDigCorr IS
BEGIN
  Process_ADCDigCorr: PROCESS (clk,reset)
    VARIABLE adc_m : std_logic_vector (24 DOWNTO 0):= adc_in;
  BEGIN
    IF reset = '1' THEN
      adc_corr <= "0000000000000";     -- mask out adc_in:
    ELSIF clk'event AND clk='1' THEN   -- cell 0 => keep adc_in(7  downto 0)
                                       -- cell 7 => keep adc_in(21 downto 0),...
      CASE cell is
        WHEN "0111"=> adc_m := "000" & adc_in(21 DOWNTO 0);      -- mask out 1st MSB cell
        WHEN "0110"=> adc_m := "00000" & adc_in(19 DOWNTO 0);    -- 1st & 2nd MSB cell
        WHEN "0101"=> adc_m := "0000000" & adc_in(17 DOWNTO 0);  -- etc...
        WHEN "0100"=> adc_m := "000000000" & adc_in(15 DOWNTO 0);
        WHEN "0011"=> adc_m := "00000000000" & adc_in(13 DOWNTO 0);
        WHEN "0010"=> adc_m := "0000000000000" & adc_in(11 DOWNTO 0);
        WHEN "0001"=> adc_m := "000000000000000" & adc_in(9 DOWNTO 0);
        WHEN "0000"=> adc_m := "00000000000000000" & adc_in(7 DOWNTO 0);
        WHEN OTHERS=> adc_m := adc_in;   -- no mask at all, only digital correction
      END CASE;                          -- v-- intercellular addition: LSB+MSB
                                         -- v-- fulladder with carry
      adc_corr <= (adc_m(24) & adc_m(22) & adc_m(20) & adc_m(18) & adc_m(16)
                 & adc_m(14) & adc_m(12) & adc_m(10) & adc_m(8)  & adc_m(6)
                 & adc_m(4)  & adc_m(2)  & adc_m(0)) +
               ('0' & adc_m(23) & adc_m(21) & adc_m(19) & adc_m(17) & adc_m(15)
                    & adc_m(13) & adc_m(11) & adc_m(9)  & adc_m(7)  & adc_m(5)
                    & adc_m(3)  & adc_m(1));
    END IF;
  END PROCESS Process_ADCDigCorr;
END Arch_ADCDigCorr;
```

10.3 List of figures

Figure 1: SNAP satellite....6
Figure 2: CCD shift registers....7
Figure 3: CCD charge coupling....7
Figure 4: CRIC-II overview....8
Figure 5: CRCIC-II photo....9
Figure 6: Noise sources....11
Figure 7: Flicker noise....11
Figure 8: CCD signal & double correlated sampling....13
Figure 9: ADC speed vs. resolution....15
Figure 10: Pipeline ADC with 1bit per stage....16
Figure 11: Transfer function 1bit cell....17
Figure 12: Transfer function 1bit cell * offset....17
Figure 13: ADC missing codes....18
Figure 14: structure n-bit ADC....18
Figure 15: CRIC-II 13bit Pipeline structure....20
Figure 16: Transfer function 1.5bit cell....21
Figure 17: Digital Correction....22
Figure 18: Calibrated output Dout up to stage n-1 with G=2....22
Figure 19: Calibration with G<2 resulting in Missing Code....23
Figure 20: Calibration with G>2, normal operation....24
Figure 21: Calibratable cells....24
Figure 22: Transfer points to measure....25
Figure 23: Calibration 1st step....26
Figure 24: Calibration 2nd step....26
Figure 25: CRIC-II signals....27
Figure 26: CRIC-II test board setup....31
Figure 27: Avnet Spartan-IIE kit....32
Figure 28: CRIC-II data flow....33
Figure 29: CRIC-II timing diagram: Start/stop of conversion cycle....34
Figure 30CRIC-II timing diagram: Readout clock and data acquisition....35
Figure 31: CRIC-II timing diagram: Zoom area....35
Figure 32: CRIC-II timing diagram: Delay between signals....36
Figure 33: 1.5bit cell schematic and signal timing of phi1/2....36
Figure 34: State chart overview of all FSMs 39
Figure 35: State chart SST(main) - Detect command....40
Figure 36: State chart SST - Run pattern continuously....41
Figure 37: State chart SST - Run pattern once....41
Figure 38: State chart SST - Load pattern....42

Figure 39: State chart SST - Stop pattern....43
Figure 40: State chart SST - Send version....43
Figure 41: State chart SST - Record config data from UART....44
Figure 42: State chart SST - Start calculation of calibration constants....45
Figure 43: State chart SST - Switch off calibration....46
Figure 44: State chart SST - Switch to raw data mode....46
Figure 45: State chart SST - Send UART strobe / acknowledge....47
Figure 46: State chart PST - Generate control pattern for CRIC-II....48
Figure 47: State chart CST - Send configuration data to CRIC-II....48
Figure 48: State chart DST - Calculation of calibration constants....49
Figure 49: State chart FST - Applying the Digital Calibration....50
Figure 50: Integral Non-Linearity....52
Figure 51:Differential Non-Linearity....53
Figure 52: INL measurement uncalibrated....54
Figure 53: INL measurement calibrated....54
Figure 54: DNL measurement....55
Figure 55: Noise measurement....56
Figure 56: Standard deviation of calibration constants....57
Figure 57: Schematic of Digital Correction....74

10.4 List of Tables

Table 1: ADC comparison....14
Table 2: Therm2Bin transcoding....21
Table 3: CRIC-II configuration register....25
Table 4: Measurement sequence....25
Table 5: Calibration decision table....26
Table 6: Comparison of signal standards....30

10.5 Further information

See my website at http://www-eng.lbl.gov/~mredelst/

10.6 Glossary

Aliasing

Visible artifacts resulting from sampling a function with a frequency less of half the highest frequency (the Nyquist frequency) present in the transfer function (e.g. of an ADC).

ASIC

Application **S**pecific **I**ntegrated **C**ircuit

CCD

Charge **C**oupled **D**evice, see *chapter 1.2*

CDS

Correlated **D**ouble **S**ampler, see *chapter 2.2*

CRIC

CCD **R**eadout **IC**, *see chapter 1.3*

DNL

Differential **N**on **L**inearity, see *chapter 8.1.2*

Digital Correction

see *chapter 5.1*

Digital Calibration

see *chapter 5.2*

Finite State Machine

A finite state machine (**FSM**) is an abstract machine that has only a finite, constant amount of memory. The internal states of the machine carry no further structure. It can be represented using a state chart. There are finitely many states, and each state has transitions to states. There is an input string that determines which transition is followed. Some transitions may be from a state to itself (loop).

FPGA

Field **P**rogrammable **G**ate **A**rray:
An integrated circuit using a logic network (gate array) that can be programmed after the device is manufactured. An FPGA consists of an array of logic elements, either gates or lookup table RAMs, flip-flops and programmable interconnect wiring.

Gray Code

A binary sequence with the property that only one bit changes between any two consecutive elements. Therefore the code has a Hamming distance of one.

Hamming distance

The Hamming distance is the number of positions in two strings of equal length for which the corresponding elements are different. It measures the number of substitutions required to change one into the other. It was named after Richard Hamming.

INL

Integral **N**on **L**inearity, see *chapter 8.1.1*

LVDS

Short for **L**ow **V**oltage **D**ifferential **S**ignaling, a low noise, low power, low amplitude method for high-speed (gigabits per second) data transmission over copper wire (*see chapter 6.1.1*).

LVTTL

Low Voltage Transistor-Transistor Logic, common I/O standard for chip to chip interfaces.

Mask

A bit pattern used to control the retention or elimination of portions of another bit pattern by using the AND or OR operation.

Mealy machine

A Mealy machine is a finite state machine where the outputs are determined by the current state and the input. In contrast, the output of a Moore finite state machine depends only on the current state and does not depend on the current input.

Nyquist zone

A Nyquist zone defines the frequency range of a signal that can be sampled by an ADC operating at a specific sampling rate.

RTL

Register **T**ransfer **L**evel -- A level of abstraction for HDL code; it generally indicates that the HDL code is synthesizable.

Synthesis

Process of compiling and building a logic representation from an HDL code on hardware level using logic gates (→FPGA).

Thermometer Codes

A binary sequence with the property that the bits are switched on sequentially just like the temperature expands the liquid in an analog thermometer:
0000 → 0001 → 0011 → 0111 → 1111

UART

Short for **U**niversal **A**synchronous **R**eceiver-**T**ransmitter, the UART is a computer component that handles asynchronous serial communication. Every computer contains a UART to manage the serial ports, and some internal modems have their own UART.

UML

Unified **M**odeling **L**anguage is a non-proprietary, third generation modeling and specification language. It can be used for modeling hardware (engineering systems) and is commonly used for business process modeling and organizational structure modeling. UML is an open method used to specify, visualize, construct and document the artifacts of an object-oriented software-intensive system under development. It's ideal for use with state charts.

VHDL

VHSIC **H**ardware **D**escription **L**anguage, VHSIC=Very High Speed Integrated Circuit

VHDL was originally developed at the behest of the US Department of Defense in order to document the behavior of the ASICs that supplier companies were including in equipment. That is to say, VHDL was developed as an alternative to huge, complex manuals which were subject to implementation-specific details.

10.7 Acknowledgments

The work described in this thesis has been developed in the time between April and October 2004 in the Lawrence Berkeley Labs. The thesis was published in January 2005. Among many other people who inspired me before and during this time I particularly want to thank some people for supporting me:

Thanks to **Prof. Dr. Bantel** for making the contact to the LBL and for sending me there. Thanks to **Chris Bebeck** for being patient when explaining new matter and for many jelly beans. Thanks to **Maximilian Fabricius** for being a smart colleague, a very good friend and for joining me in being coffee addicted. Thanks to **Armin Karcher** for introducing me to the Lab and the Bay Area and for providing guidance. Thanks to **William Kolbe** for having great, inspiring and relaxing talks and for spending unforgettable times in San Francisco. Thanks to the **LBL** for being an excellent place to work and for giving me the SPOT Award for my work. Thanks to **Henrik von der Lippe** for an excellent project management and for sharing his knowledge. Thanks to **Julia Lundy** for improving my English and for having a fabulous time. Thanks to **Joao Pequenao** for sharing numerous relaxing hours, for cracking many stale jokes and for making people eat things they didn't want to eat. Thanks to **Prof. Dr. Sapotta** for being a very good professor at the University of Applied Sciences in Karlsruhe and for giving motivation to keep on studying. Thanks to **Jean-Pierre Walder** for making the link to the core of CRIC-II (I know you were right!) and for supporting Max and me while debugging the design. *You all made it possible!*

10.8 References

[1] Yun Chiu, Cheongyuen W. Tsang, Borivoje Nikolic´, Paul R. Gray
Least Mean Square Adaptive Digital Background Calibration of Pipelined ADCs
IEEE Transactions on Circuits and Systems: Vol. 51, No. 1, January 2004

[2] S.Y. Chuang, T. L. Sculley
A Digitally Self-Calibrating 14-bit 10-MHz CMOS Pipelined A/D Converter
IEEE JSSC, Vol. 37, No. 6, June 2002

[3] Mark Ferriss, Joshua Kang
A 10 bit 100MHz pipeline ADC
University of Michigan, 598 design project, 2004

[4] R.I. Hornsey
Noise in Image Sensors
University of Waterloo, March 1999

[5] G. R. Hopkinson, D. H. Lumb
Noise reduction techniques for CCD image sensors
X-ray Astronomy Group, University of Leicester, June 1982

[6] Walt Kester, James Bryant
ADC Architectures
Analog Devices, March 2004

[7] Walt Kester
Digital Video and Display Electronics
Analog Devices, March 2004

[8] Walt Kester
Sampling Theory
Analog Devices, March 2004

[9] Walt Kester
Testing ADCs
Analog Devices, March 2004

[10] Hae-Seung Lee
A 12-b 600ks/s Digitally Self-Calibrated Pipelined Algorithmic ADC
IEEE JSSC, Vol. 29, No 4, April 1994

[11] Benoit Provost, Edgar Sánchez-Sinencio
A Practical Self-Calibration Scheme - Implementation for Pipeline ADCs
IEEE Transactions on Instrumentation and Measurement, Vol. 53, No. 2, April 2004

[12] Defining the Specifications
Dan Sheingold, Walt Kester
Analog Devices, March 2004

[13] Jean Pierre Walder
A 13bit Pipeline ADC
Lawrence Berkeley Lab, February 2004

[14] Jean Pierre Walder
The 1.5bit cell
Lawrence Berkeley Lab, January 2004